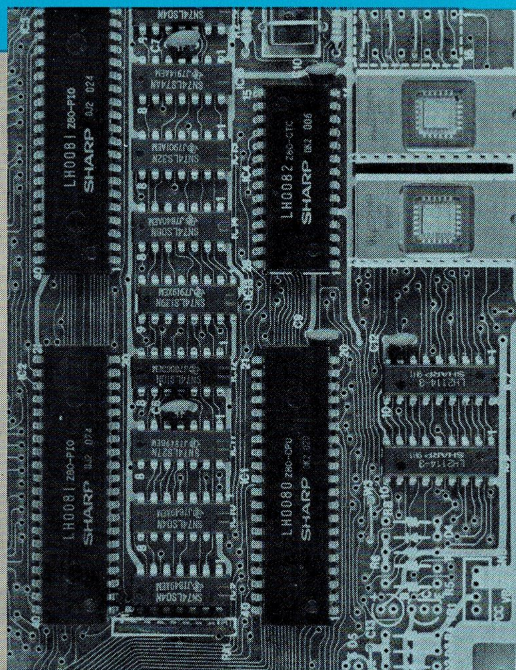
図解 マイクロコンピュータ

Z-80の使い方

横田英一 著
オーム社



◆関連図書案内◆

図解 初めて マイコンを学ぶ人のために

(A 5 判 170頁)
鎌田信夫・野島晋 監修　大原茂之 著
初めてマイコンを学ぼうとする人達を対象に，その機能と使い方の入口までを，やさしく解説．

図解 マイコンの使い方

(A 5 判 188頁)
小牧常松・大條廣 共著
ハードウエア構成の概要と，ソフトウエアの構成の仕方，さらに近い将来一般技術者にも必要となるOSの概念について，やさしく解説．

図解 マイコンのための BASIC入門

(A 5 判 180頁)
小牧常松・大原茂之 共著
実際にプログラミング技術を会得するために，特にマイコン用プログラミング言語として目覚ましく普及しているBASICを，例題を豊富に挿入し，わかりやすく解説．

図解 マイコンのための アセンブラ入門

(A 5 判 160頁)
大原茂之・倉田了一 共著
最も一般的なインテル8080系に基づいて，アセンブリ語とアセンブラの使い方について，初学者にもわかるようやさしく解説．

図解 マイコンのための インタフェース入門

(A 5 判 180頁)
大原茂之・北沢国男 共著
インタフェースの基礎概念からアナログ量をマイコンに取込むまでの実際を図解により平易に解説．

図解 マイクロコンピュータ

Z-80の使い方

横田英一 著

オ ー ム 社

すいせんの言葉

私達のまわりには，今大きな変革が起りつつあり，しかも猛スピードで，さらに加速されつつ事は進んでいます．

19世紀に起った第一次産業革命が，肉体労働の軽減を機械化で行ったように，現在の姿は頭脳労働の軽減や人間性の回復を電子技術を中核として実現されつつあります．特に2進法の数理から生れた4ビットのマイクロコンピュータは8ビットに飛躍し，さらに必要性により16ビット系に拡大しようとしております．

最も手近で最も多くの用途があり，汎用性の高い8ビット系がマイクロコンピュータの中心として，オフィスオートメーション，ホームコンピュータ，インテリジェンス機器等の主要電子部品の核であることは間違いないと信じます．

8ビット系マイクロコンピュータの中でも本書がご案内するZ-80は，ハードウェア的にもソフトウェア的にも，きわめて高度なマイクロコンピュータであり，アーキテクチャでも命令体系でもミニコンピュータに匹敵し，時にはしのぐほどになっています．

Z-80の命令体系はインテル社の8080Aの命令をすべて含み，容易に置換えられ，そのうえ8080にない多くの新しい命令やアドレス指定方式が加えられています．

たとえば文字列の検索は準備操作によって1命令で実行でき，これは他のものの4命令と等価になっています．

さらにZ-80には，その能力を補完する周辺デバイスとして，PIO, SIO, CTC, DMA等のファミリーが用意されています．

世間には数多くの良書が出ていますが，本書はわかりやすい手ほどきのものとして，ご愛読，ご再読をおすすめします．

1981年4月

シャープ株式会社
取締役　經澤崇泰

はしがき

マイクロコンピュータがエレクトロニクス分野に画期的な進歩をもたらしたのも，早くも過去の語り草になろうとしております．すでにマイクロコンピュータは多くの応用製品を生み出し，研究熱心なユーザによって秘められた能力をさらに深く追求されて，あらゆる分野に恩恵をもたらしております．

今後の電子技術の一翼としてマイクロコンピュータ応用技術が重要な位置を占めるであろうことは自明です．学校や企業の実験室には，テスタやシンクロスコープと並んでプログラム開発用の装置が導入され，資料室には回路図といっしょにプログラムが保管されることになります．

単なる時流やはやりものではない，根底的な変遷の中にあって，マイクロコンピュータ応用技術の取得は，真空管がトランジスタにおき変った以上に早いスピードで必要性を増しております．

マイクロコンピュータは，1971年にインテル社から発表されたI 4004以来，10年間の間に数回の世代交替がありました．この間に機能は飛躍的に拡大し，処理速度，プログラムサイズ，部品構成，応用技術等の面で数段の進歩をとげてきました．8ビット系では，そのほぼ頂点にあるのがザイログ社のZ-80ファミリです．8ビット系でこれ以上の機能を要求すれば16ビット系へ移行するのが一般的な考え方ですが，最も多くの用途があり汎用性の高い8ビット系は今後もマイクロコンピュータの中心にあることに変わりないはずです．

本書では，エレクトロニクス技術の研さんに励む学生諸氏，および企業にあってマイクロコンピュータ導入を目前に技術修得を課題とする電子技術者諸氏のために，今後も主流の地位を他に譲ることのないであろうZ-80を例に，動作原理から応用例までを解説したものです．本書でZ-80をマスタしてしまえば，どんなコンピュータでも容易に理解できると信じます．

マイクロコンピュータは，ハード面，ソフト面が有機的に結合し合いシステム

を形成していますので，ページに従った展開による一次元的な説明では到底表現しきれないものです．平面的，二次元的なつながりを重要視する意味で項目間の参照は不可欠です．また，一読して理解できないことを嘆くなら再読を勧めます．1回目ではぼんやりと全体像をとらえ，2回目では後に書いてあることを思い起こしながら前の説明を見ていきます．次に自分でプログラムを考えてみて下さい．例題のプログラムを自分なりに手なおしするだけでもよいのです．自分自身で試行することにより不明点をとらえ，解説を読みなおせば確実に身につきます．マイクロコンピュータは物性や自然科学とは違って，人間が考え出して作ったものです．使いやすく，覚えやすくするためにさまざまな知恵をしぼってあります．自然現象の探究とは異なった考え方で対処すれば，いたって気軽に入っていけるものと思います．

本書では，基礎的な項目について詳細な解説は，他に良書が多数あることを理由に，ふれる程度にとどめた点もあります．オームの法則に始まる電子回路，ブール代数，論理素子（TTL）についての最低限の知識は必要です．また，マニュアルにある正確緻密な表現よりも，平易な解説を心がけました．用語がわからないだけでコンピュータは理解できない難物と思ってしまうことも多く，この点も読み進むうちにだんだんとなじめるように配慮したつもりです．

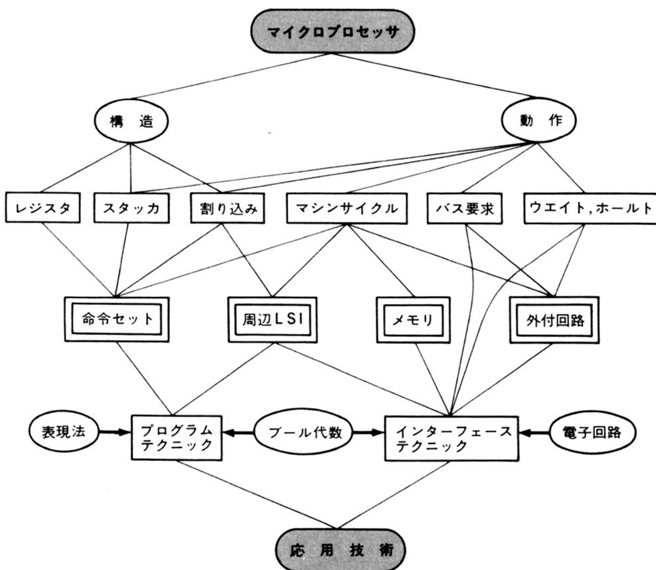
変貌する電子技術の中にあって，新しい分野への進出を目ざす読者諸兄の勉学の一助となり，エレクトロニクス産業界の発展のためのごく小さな後押しになれば筆者の意とするところであります．

最後に，執筆にあたり御理解と手間をいとわない御協力を賜わったシャープ株式会社の関係部課の皆様，オーム社出版部の方々に心から感謝の意をささげる次第であります．

1981年 4月

著者しるす

マイクロプロセッサの応用技術



基礎知識

- 表現法
  - （用語）
  - （16進表現）
  - （アセンブリ言語）
- ブール代数
  - （真理値表）
  - （TTL ロジック）
- 電子回路
  - （オームの法則）
  - （電子部品）
  - （半導体）

付加事項

- 経験と知識に基づく　慣れ≒ヤマカン　を養うこと．
  自分でやってみることが重要
- 中身がどうなっているかは問題ではない．
  どうすればどうなるかを知って使いこなすことが重要．

目　　　　次

1　マイクロプロセッサの特質……2

2　マイコンシステムの種類……4

3　特質を活かすシステム設計……6

4　Z-80　ファミリの特徴……8

5　Z-80　CPU……10

6　Z-80　PIO……12

7　Z-80　CTC……14

8　Z-80　DMA……16

9　Z-80　SIO……18

10　メモリの種類と用途……20

11　ビットパターンと16進表現……22

12　プログラムの実行……24

13　CPUの信号のやりとり……26

14　データバス，アドレスバスとシステム制御信号……28

15　命令語の構成……30

16　命令の実行……32

17　アセンブラ記法のルール　-1-……34

18　アセンブラ記法のルール　-2-……36

19　命令の分類……38

20　メモリ空間とIO空間……40

21　アドレスデコーダ……42

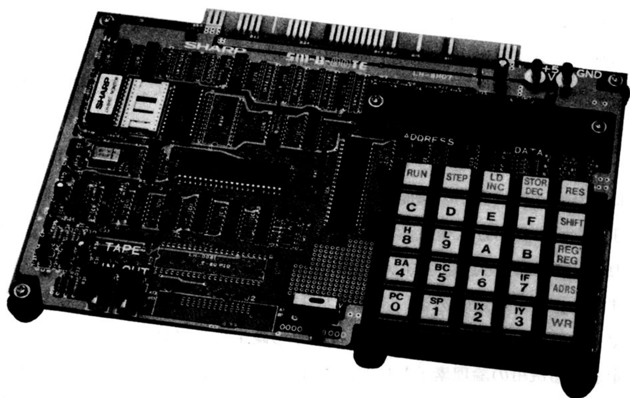
22　バスバッファ……44

23　クロック……46

24　リセット……48

25　システム構成……………………………………………………………50
26　フェッチサイクルの動作……………………………………………………52
27　メモリリードサイクル………………………………………………………54
28　メモリライトサイクル………………………………………………………56
29　IO リードサイクル，IO ライトサイクル……………………………………58
30　リフレッシュサイクル………………………………………………………60
31　CPU と周辺の接続（インターフェース）……………………………………62
32　ウェイト信号とホルト信号…………………………………………………64
33　割り込みの概念……………………………………………………………66
34　ノンマスカブルインタラプト（$\overline{\mathrm{NMI}}$）………………………………………68
35　インタラプト（$\overline{\mathrm{INT}}$）…………………………………………………………70
36　モード0のインタラプト……………………………………………………72
37　モード1のインタラプト……………………………………………………74
38　モード2のインタラプト……………………………………………………76
39　デージーチェーン……………………………………………………………78
40　バス要求と応答………………………………………………………………80
41　内部レジスタの構成…………………………………………………………82
42　A, I, R, F レジスタ…………………………………………………………84
43　汎用レジスタ…………………………………………………………………86
44　補助レジスタと交換命令……………………………………………………88
45　IX，IY レジスタ………………………………………………………………90
46　スタックとスタックポインタ（SP レジスタ）……………………………92
47　転　送　命　令………………………………………………………………94
48　算術演算命令…………………………………………………………………96
49　論理演算命令…………………………………………………………………98
50　ビット操作命令………………………………………………………………100
51　ローテート，シフト命令……………………………………………………102
52　ブロック転送，ブロックサーチ，ブロック入出力命令……………………106
53　ジャンプ命令…………………………………………………………………110

54　コール，リスタート，リターン命令（サブルーチン）……114
55　Fレジスタとフラグ変化……116
56　2進化10進数と10進補正命令……118
57　ペリフェラルのプログラミング……120
58　PIO モード 0 の動作……122
59　PIO モード 1 の動作……124
60　PIO モード 2 の動作……126
61　PIO モード 3 の動作……128
62　PIO のプログラミング……130
63　PIO のプログラム例……132
64　CTC カウンタモード……134
65　CTC タイマモード……136
66　CTC のプログラミング……138



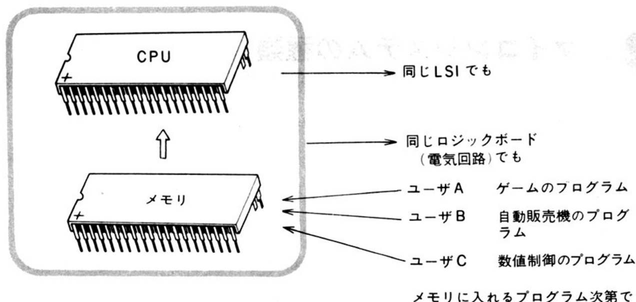
ワンボードコンピュータ製品例　SM-B-80TE（シャープ）

1 マイクロプロセッサの特質

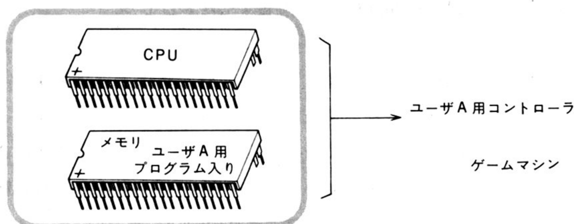
コンピュータという機械が発明された目的は，人間には不可能な複雑な計算を短時間に行なうことでした．たとえば円周率の計算を一生の大半を費やして行なっても数百桁だったのに，コンピュータなら瞬時にもっと多くの桁を間違いなくやってのけることができます．人間を月へ送り込むのも，正確な軌道計算が刻々と得られるためにできたことで，コンピュータなしでは考えられないのです．ある手順の決まった仕事を，高速かつ正確に処理することのできる機械，これがコンピュータの目的であり，その目的は十分に達せられたといえるでしょう．

ところが，ストアド プログラム方式すなわち，処理手順を示すプログラムを，コンピュータに入力すべき情報の一つとして扱うことのできる今日のコンピュータ方式では，当初の目的以外に，大きな特徴が認識されました．全く同一の機械（ハードウェア）に対し，使用者が作成するプログラム（ソフトウェア；これは使用者によってそれぞれに異なる）をメモリに入れることにより，使用者のそれぞれの目的に合った仕事をする，という特徴です．コンピュータ メーカは同じ機械を大量に生産すれば，科学計算であろうと，在庫管理であろうと，人事管理，生産ラインコントロール，座席予約，相性判断まで，プログラム次第で適用されてしまうのです．

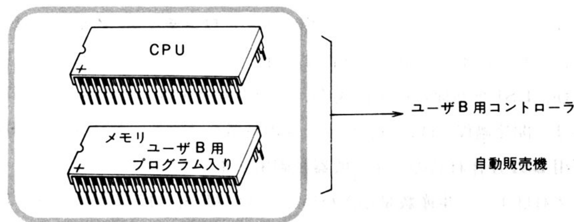
半導体の技術が進歩して数千，数万の部品を数ミリ四方の中へ作り込めるようになったとき，専用の機能を持ったLSIが，さまざまな目的に合わせて作られました．ところが大量生産にしか向かないLSIの弱点を補うべく，半導体メーカは汎用LSIの思想をコンピュータに見出したのです．コンピュータをLSI化すれば，多く用途のある，つまり大量に売れるLSIが作れると考えたのです．マイクロ プロセッサはこのような中から誕生し，プログラムによって機能が決定される汎用の論理素子として，すばらしい発展をとげ，現在も発展しつつある，“電子部品”なのです．マイクロプロセッサを中心としたマイコン システムを利用しようとする場合，この辺を的確に把握してかかることが重要です．



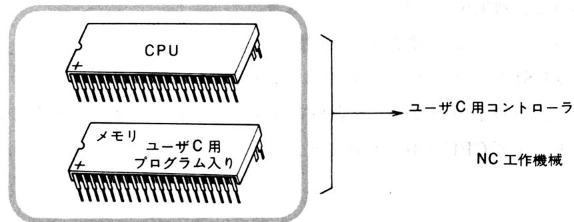
メモリに入れるプログラム次第で



ゲームマシン



自動販売機



NC工作機械

2 マイコンシステムの種類

マイコンと呼ばれるコンピュータ システムには，いくつか考えられます．一つに，パーソナル コンピュータやオフィス コンピュータに近い性能を持つものを“**マイコン**”と呼びます．これらは，コンピュータを小さく，安価にまとめ，事務所や大学の研究室や個人の知的玩具としてまで普及した個人用コンピュータ システムです．電気の知識がなくても，簡単な言葉を覚えれば誰でも使えるテレビ，ステレオなどと肩を並べる全く新しいマニアライクな道具といえます．この場合のマイコンは，“**My Computer**”と解釈すべきでしょう．

もう一つの“マイコン”は，パソコンほどのはなばなしさはありませんが，工作機械や自動販売機や家電製品に組み込まれている，機器制御用の**ワンボードマイコン**です．基板上に作られたマイクロ コンピュータ システムは，産業用としての中心をなすものです．各種の用途でそれぞれの目的に合った設計がなされ，特徴を出しています．組み込まれた時点では，プログラムは固定化され，機器のユーザには，コンピュータとしての使用はできないのが普通です．

1個のLSI上にメモリや入出力ポートを作り込んだ**ワンチップ マイコン**は，時計や電卓に使われています．中身を分析すれば，マイクロ コンピュータに違いありませんが，LSIを外面から見た場合，一つの機能を持った専用LSIになってしまいます．開発過程では，コンピュータの特徴が活きて，短時間に安価な開発経費で専用LSIが作れるのです．機器制御用マイコン システムをワンチップ化したと考えればよく，生産数量がきわめて大きい場合，コストやサイズ，消費電力などの点で，効果を上げることができます．

また，マイコン システムを構成する最重要部品であるマイクロ プロセッサ，つまりCPUのLSIを“マイコン”と呼びます．しかし，これだけではコンピュータとはいえず，周辺やプログラムを含めてコンピュータ システムが完成するのですから，あくまでCPUと呼ぶべきです．

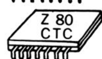
ワンボード・マイコンやパーソナル・コンピュータはマイコンシステムの応用製品



マイクロプロセッサ
(CPU)

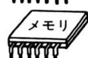


入出力LSI

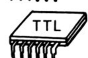


カウンタタイマなど

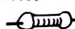
周辺LSI



メモリ



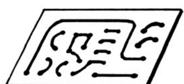
TTLIC



抵抗

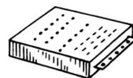


コンデンサなど



基板
コネクタ
など

ワンボード
マイコン
(制御用)



電源

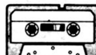


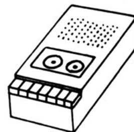
ブラウン管
プリンタ
など



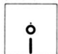
キーボード

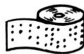
ソフトウェア





カセットテープ
レコーダ
フロッピ装置
など



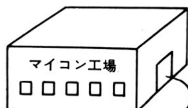


パーソナル・コンピュータ

一方，ワンチップマイコンは開発過程がコンピュータである，専用LSIである



プログラム





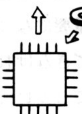
電卓用LSI

電卓工場

液晶
表示板

1234

電卓用
LSI



電池

キーボード

1	2	3	C
4	5	6	√
7	8	9	%
+	−	×	÷

3 特質を活かすシステム設計

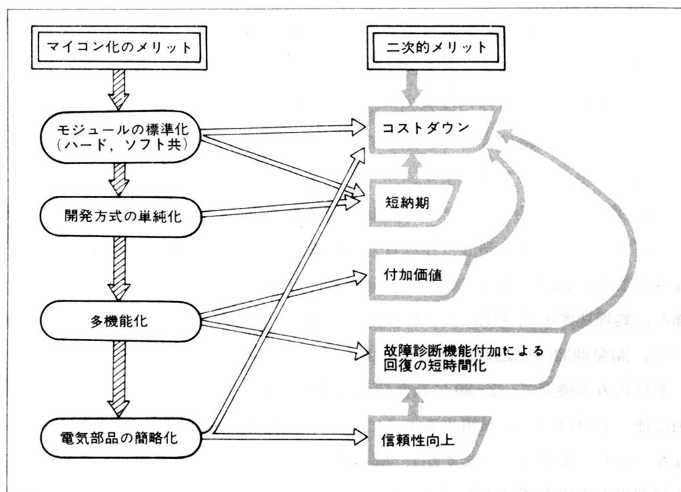
コンピュータは，仕事を高速に処理することと，同じ機械（ハードウェア）でもプログラム（ソフトウェア）を変えるだけで異なった働きをさせられること，の2点が最大の特徴です．制御用マイコンは，メカニズムやリレーや論理回路によるワイアード ロジックからの延長線上にあり，高速性では，TTL ロジックには一歩ゆずります．むしろ，プログラムにかかる機能面の柔軟性が大きなメリットです．また，多機能化することが比較的容易であり，故障時に自己診断をさせることすら可能です．マイコン ボードとして決定された能力の限界以内であれば，同一の規格化された基板を何種類もの製品に応用することができ，開発コストの引き下げに効果が期待できるようになります．プログラムもモジュール化し保管をすることにより，経験の蓄積ができ開発期間の短縮がはかれます．

マイコン システムの設計にあたっては，ハードウェアとソフトウェアの分担を決定することがポイントになります．回路設計とプログラム設計の前段階としてのシステム設計といわれる作業です．往々にして，電子技術者が設計したシステムはハードにウェイトがおかれてしまう傾向があります．ハード，ソフト共に精通した人が，目的の機能と，開発コスト，ランニング コスト，メンテナンス コストなどすべてにわたって検討して決定すべきです．特に部品やボード，プログラムの標準化という意味から，将来にわたって要求される機能の変動を見通すことが重要になります．変動要素はソフトウェアの分担とし，メモリの交換だけで対応できなければなりません．

概してコンピュータの特質を考えるならば，タイミング的に不可欠な部分をハードウェア化し，他はソフトウェアで実現する方法に利があるといえます．ハードウェアを最小限に止めることは，材料費，経年劣化，故障率を低減する上で有利であることは明白です．

マイコン適用分野

- 多機能であって，ICレベルの論理素子で構成すると，大規模になりすぎるもの ──→ パーソナル コンピュータ
- 大同小異の機種が多く，いちいち論理素子で構成したのでは設計に手間がかかりすぎるもの ──→ 端末装置，NC 機器
- 納入先ごとの変更があるもの ──→ ビル防災システム，ホテル管理システム
- あとで変更のあり得るもの ──→ 自動販売機，料金計算機
- 開発時間の限られる場合 ──→ ハードウェアは標準品を使い，プログラムだけを入れ換える



必要条件

改善意欲 ──→ 社内規格の統一・コストダウン

先行投資 ──→ 開発装置購入

技術力（技術吸収力）──→ 要員教育

4 Z-80 ファミリの特徴

これまでに最も普及したマイクロ プロセッサにインテル社の8080系があります．日本でも各社がセカンド ソースを発表しています．

8080系は，　8008→8080→8080A→8085　と進化したのですが，この流れの延長線上に全く新しい考え方をとり入れて設計されたのが，ザイログ社のZ-80です．したがって8080系の機能はすべて引き継いでおります．

Z-80は，ハード的にはできるだけ少ない部品で構成できるようになっています．ファミリは5種類しかありませんが，これらのLSIに機能を分散させてあります．ファミリを構成するLSI間のインター フェースは，多少複雑ですが，Z-80の設計者の意図からはみ出さない限り問題になることはありません．むしろ，部品点数の削減に大きく貢献しています．クロックは単相の方形波でよく，電源は5ボルト単一で働きます．

8080系の持つ命令は，すべてZ-80の命令群の中に含まれていて，マシン語も同じです．アセンブリ語は，かなり差異がありますが，整理統合されており，大変覚えやすい言語になっています．命令数はきわめて多く，プログラム容量の短縮と，処理速度の高速化に大いに役立っており，わかりやすい言語体系とあいまって，開発期間の短縮が期待できます．

割り込みの機能で，Z-80に新たに追加されたモードは，メモリ アドレスを自由に使ってプログラムが組めるようになっています．また周辺のLSIが何個つながっても（限度はありますが），標準的な接続で割り込み処理ができます．特に制御用の応用面では重宝です．

総合的に，Z-80は8ビットのプロセッサとしては，いまのところ最上位の機能を持ち，将来もこれに代わるものは必要とされないだろうとさえいわれています．もっと高機能が望まれるときは，CPUを複数にして分散処理をするか，16ビット以上の機種を使用するほうが，8ビット・プロセッサにさらに高度な機能を要求するより自然な流れではないでしょうか．

Z-80ファミリ

- Z-80 CPU (Central Processing Unit)
  - Z-80 PIO (Parallel Input/Output Interface Controller)
  - Z-80 CTC (Counter Timer Circuit)
  - Z-80 DMA (Direct Memory Access)
  - Z-80 SIO (Serial Input/Output Interface Controller)

特 徴

- ＋5V単一電源 ──→ TTLコンパチブル
- 単相クロック ──→ クロックジェネレータ不要
- 割り込み機能 ──→ コントローラ不要．プログラムが簡単
- レジスタ群 ──→ プログラム容量の圧縮・スピードアップ
- 命令セット ──→ プログラム容量の圧縮・スピードアップ
- リフレッシュ機能 ─→ ダイナミックメモリ使用可能 ─→ コストダウン

命令セットの特徴

- 16ビットの引き算
- レジスタ，メモリのビットのセット，リセットテスト
- メモリブロックの転送，サーチ，入出力
- 2の補数をとる命令
- 4ビット単位のローテーション
- 整理されたニモニック（暗記用命令コード）

5 Z-80 CPU

CPU (Central Processing Unit) はZ-80 ファミリの中核をなすものであり，ファミリの性格を決定する中心要素です．なお，ファミリでCPU以外のLSIを，周辺LSIとかペリフェラルと呼ぶことがあります．

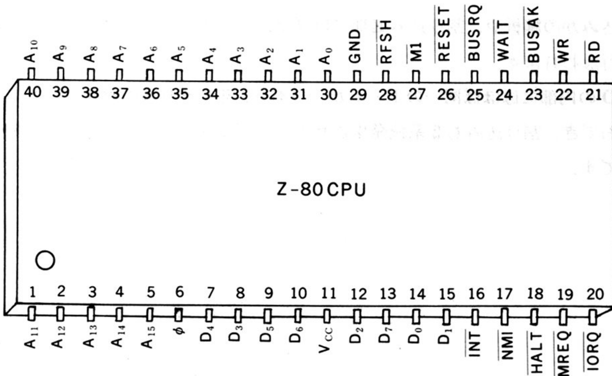
Z-80 CPUには8ビットの汎用レジスタが12あります．組み合わせて16ビット レジスタとして使用することもできます．インデックス レジスタは2個あり，他にスタック・ポインタ，プログラム カウンタ，割込ベクトル レジスタ，メモリ リフレッシュ レジスタ，フラグ レジスタ，アキュムレータがあります．

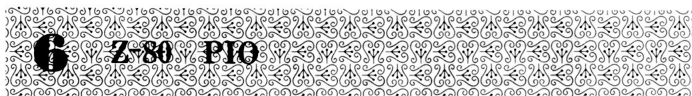
これらのレジスタや命令を解析し実行する制御部や演算部の働きは，以下の項で解説します．

外部端子には，アドレス バス16本，データ バス8本，システム制御線6本，CPU制御線5本，バス制御線2本，それにクロックと電源が出ています．図で，トライ ステートと記入されている端子は，"H"と"L"の二値状態のほかに，端子と内部が電気的に切り離された“ハイ インピーダンス”状態を持つものです．動作と関係のないときはこの状態になっており，外部からの負荷にならないようになっています．

クロックは単相でよく，通常2.5MHz以下，Aバージョンは4MHz以下で働きます．通常のバージョンとAバージョンはクロックの最大値が異なりますが，他の機能や電気的特性は共通です．以下のファミリにもすべて通常のものとAバージョンがあります．同じバージョンのLSIを揃えて使わなければ意味がありませんが，AバージョンのLSIを2.5MHz以下で使用しても問題はありません．

命令解析部
演算部
レジスタ群
制御部
データバス制御部
アドレスバス制御部
φクロック
V_{CC} (+5V)
GND
$\overline{M1}$
$\overline{MREQ}$ Ⓣ
$\overline{IORQ}$ Ⓣ
$\overline{RD}$ Ⓣ
$\overline{WR}$ Ⓣ
$\overline{RFSH}$ Ⓣ
システム制御線
$\overline{HALT}$
$\overline{WAIT}$
$\overline{INT}$
$\overline{NMI}$
$\overline{RESET}$
CPU制御線
$\overline{BUSRQ}$
$\overline{BUSAK}$
バス制御線
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
A_{15} ………… A_0
データバスⓉ
アドレスバスⓉ
Ⓣ はトライステート





6 Z-80 PIO

CPU の次によく使われるのが，この **PIO**（Parallel Input/Output Interface Controller）です．CPU からの信号を受けて，外部の装置に出力したり，外部装置からの信号を受けて，CPU へ伝えたりするデバイスです．

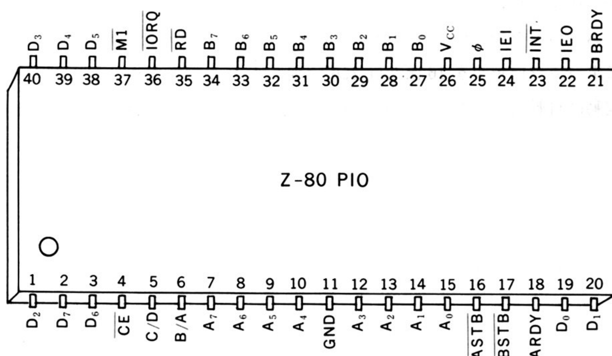
CPU の信号は，データ バスを入出力以外の用途にも時分割で共用しているため，大変複雑な信号になっています．この中からある特定の時点のデータバス上の信号をとらえて外部との受け渡しをするためのもので，一般にラッチと呼ばれるロジックに，使いやすい機能をつけ加えた LSI といえます．信号は 8 ビット並列に扱います．動作は次の四つのモードがあります．

モード 0　出 力モード
モード 1　入 力モード　　ハンドシェークにより 8 ビットデータを入・
モード 2　入出力モード　出力する．
モード 3　ビットモード（ビット単位に入力，出力を選択できる）

モードやその他の動作条件は CPU からの信号により内部の制御レジスタに制御ワードを書くことによって設定されます．割り込みは CPU と PIO（または他のペリフェラル）のみで制御され，優先順位の決定や，どのペリフェラルからの割り込みかの解析や，割り込み処理プログラムルーチンからのリターンは，自動的に行なわれます．

PIO の内部にはほぼ同一のポートが二つあり，それぞれ別の動作モードで使うこともでき，割り込みも 2 系統発生させることができます．優先順位は，A → B の順です．

データバス
D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7
データバス制御部
制御部
A ポートベクトル
B ポートベクトル
制御レジスタ
〃
$\overline{M1}$
$\overline{RD}$
$\overline{IORQ}$
$\overline{CE}$
B/A
C/D
$\overline{INT}$
IEI
IEO
割込み制御部
A ポート
B ポート
入力レジスタ
出力レジスタ
ハンドシェーク
φクロック
V_{CC}
GND
A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
ARDY
$\overline{ASTB}$
B_7 B_6 B_5 B_4 B_3 B_2 B_1 B_0
BRDY
$\overline{BSTB}$



7 Z-80 CTC

CTC (Counter Timer Circuit) は，パルスのカウントダウンをするLSIです．不特定周期の外部パルスをカウントし設定数になると割り込みを入れたり，ゼロカウント出力を出したりすることができます．不特定なパルスでなくクロックのような決まった周期のパルスをカウントすることにより，タイマとしても使えるわけです．

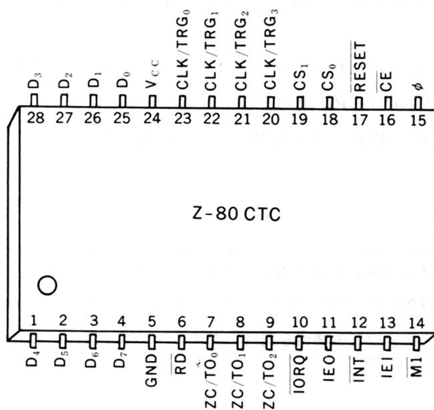
何個目のパルスで割り込みまたはゼロカウントするかの設定やカウントパルスのエッジ（立ち上りか立ち下りか）の選択などはCPUからの書き込みによってプログラムされます．またカウントの途中での残り数はCPUがカウンタの内容を読み出すことによって知ることができます．

設定できる数は1～256までですが，くり返えして何回目かの割り込みでカウント終了とすることにより範囲は広がりますし，ゼロカウント出力を次のカウンタへ入れてやれば，256^2までが扱えます．n個つなげば256^nです．

1個のCTCには四つのチャンネルが内蔵されていて，別々の目的に使用することができます．ただしピン数の関係で4個目のチャンネルは，ゼロカウント/タイムアウトの出力が出ていませんので，割り込みによりカウント終了となる使い方しかできません．

システムクロックのカウントダウンによるタイマとしての使用時は，プリスケーラによってクロックを16または256分周したパルスをカウントします．タイマの起動は自動的に行なうことも，また外部トリガによって行なうこともできます．

D0
D1
D2
D3
D4
D5
D6
D7
φ
VCC
GND
データバス制御部
制御部
ベクトル
割り込み制御部
M1
RD
IORQ
CE
CS0
CS1
INT
IEI
IEO
ch0
プリスケーラ
定数レジスタ
ダウンカウンタ
ch1
ch2
ch3
CLK/TRG0
ZC/TO0
CLK/TRG1
ZC/TO1
CLK/TRG2
ZC/TO2
CLK/TRG3



 Z-80 DMA

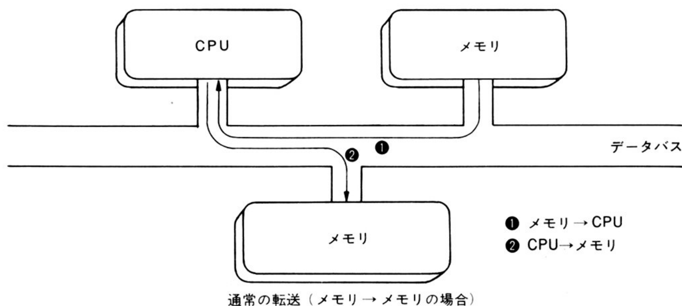
メモリ内，あるいはメモリと入出力ポートとのデータの転送は，通常はCPUが一度レジスタへ読み込み，次に書き出すという方式で行ないます．Z-80には，単命令で一連の複数のデータを転送するブロック転送命令があります．プログラムステップ数は少なくなり，便利ではありますが，転送に要する時間は，1バイトずつくり返しループを実行して転送するのと大差ありません．この転送の作業だけを高速に行なう周辺LSIが**DMA**（**Direct Memory Access**）です．

CPUから，あらかじめ転送されるデータの入った元のアドレス（ソース）と転送先のアドレス（ディスティネーション）と，転送するバイト数をDMA内のレジスタへ書き込めば，自動的に指定の転送を行ないます．ソースまたはディスティネーションは，自動的にカウントアップされますが，特定のIOポートへ次々に出力または入力するときはカウントアップを止めておくこともできます．

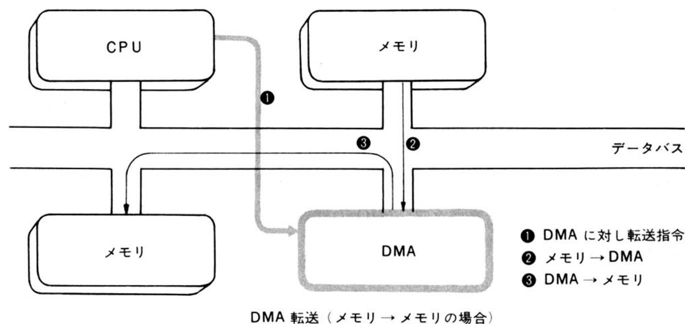
転送だけでなく，転送しながら，あるいは転送をせずに指定されたビットパターンとデータのビットパターンの一致をチェックすることもできます．これを**サーチ**と呼んでいます．サーチの終了（一致）は，転送終了とは区別できるよう割り込みがかかります．

DMAが動作中は，アドレスバスやデータバスは占有されますのでCPUは待ち状態になりますが，指定によりCPU優先として一バイト単位に転送することや，外部ロジックからの切り替えによりCPUへバスをあけ渡すこともできます．

> DMAやSIOを使うシステムは，かなり大がかりなものになります．
>
> 本書では，初歩的な内容をわかりやすく解説することが主目的ですから，これらを対象からはずし，次のステップでの修得を期待します．



通常の転送（メモリ→メモリの場合）



DMA 転送（メモリ→メモリの場合）

9 Z-80 SIO

PIOは，データを並列に入出力するためのポートコントローラですが，このSIO（Serial Input/Output Controller）は，データを直列にすなわち時間経過に従って入出力するポートコントローラです．電話回線を使って長距離の通信を行なう場合はもとより，同一機器の匡体内でもケーブルが長くなる所は，データ通信線の数が少なくてすむシリアル伝送を利用することがあります．もちろん他の条件が同一なら伝送速度はパラレルの1/8になることはやむを得ません．

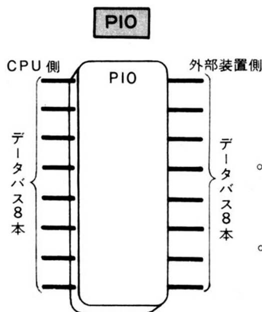
直列データ伝送の方式には，同期・非同期，バイト指向・ビット指向などがあり，また通信速度や電圧，電流など，何種かの規格があります*．コントロールプログラムと外部付加回路によってほとんどの方式に対応できる，きわめてぜい沢な機能を持ったLSIです．

このSIOのチップ（LSIの中のシリコンウェハ上の本体）は41個の外部端子をもっていますが，パッケージは40ピンであるため，外部ピンに接続されない端子があるもの2種類と，2本の端子を1本のピンに接続したものの合計3種類のSIOがあります．ボンディング・オプションと呼んでおりSIO-0，SIO-1，SIO-2と区別しています．

SIOはきわめて多機能であるため，すべての動作を理解しようとせず，シリアル伝送の方式を理解したうえで，SIOの機能の必要な部分を使用するよう考えるのが効果的です．

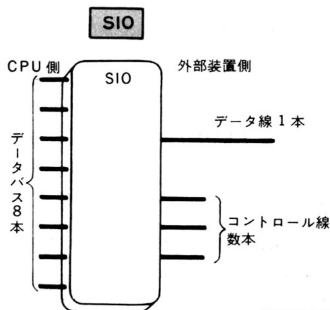
* ジョン・E・マクナマラ著：コンピュータ・データ通信技術，（CQ出版）にくわしい．

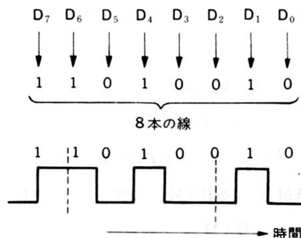
PIO



- 外部装置から送り込まれるデータバスの内容をある瞬間をとらえて（IO サイクルに）CPU 側のシステムデータバスに送り出す.
- CPU 側のシステムデータバスの内容をある瞬間をとらえて保持し，外部装置への受け渡しのタイミングをとる.

SIO





- 外部装置から送られてくるデータ線上のパルス列を 8 ビットの並列データになおし CPU 側へ送り出す.
- 8 ビットの CPU 側データバスの内容を時間と共に順次データ線へ送り出す.
- 上記に必要なコントロール信号，内容の検査をするための付加情報を扱う.

10 メモリの種類と用途

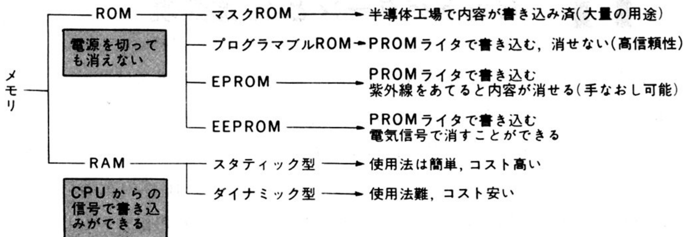
マイコンシステムで使われるメモリには何種類かありますが，現在ではほとんどが半導体素子で構成されるLSIメモリです．磁性体メモリは読み書きが自由にできて，かつ電源を切っても内容が消えないのですが，価格やドライブ回路の点であまり使われません．

LSIメモリはROM（リードオンリーメモリ）とRAM（ランダムアクセスメモリ）に大別されます．ROMは書き込みに特別な装置が必要で，CPUからのメモリライトはできません，ただし電源を切っても消えないため，プログラムを入れておくのに主に使われます．RAMは，CPUからの書き込みができますが，電源を切ると内容は消えてしまいます．

ROMには，マスクROMとPROM（プログラマブルROM）があります．マスクROMは半導体工場で生産する時点で内容が決定されてしまいますので，同一内容の量産に向いています．PROMは，ROMライタという装置で書き込みますが，一度書いたら変更できないもの（バイポーラ型PROM）と，電気信号で消去できるEEPROM（エレクトリカルイレーザブルPROM）と紫外線照射により消去できるUVEPROM（ウルトラバイオレットイレーザブルPROM）があります．一般的にはUVEPROMを単にEPROMと呼び多く使用されています．

RAMにはSRAM（スタティックRAM）とDRAM（ダイナミックRAM)があります．SRAMはフリップフロップにより構成されたメモリで使いやすい特徴があります．DRAMはコンデンサの電荷蓄積を応用したメモリですから，時間と共に消えてしまいます．消えないうちに（実際には2ms以下ごと）に一度読み出して再び同一内容を書き込むようにします．これを**リフレッシュ**といい，Z-80では，CPUが命令コードを解析している時間を利用して自動的にリフレッシュする機能を持っています．コスト的にはメリットがあるのですが，ドライブのインターフェースが複雑になりむずかしさが残ります．最近ではメモリ内部にリフレッシュ機能を持ち，疑似的にスタティックと同様に使えるダイナミックRAMもあります．

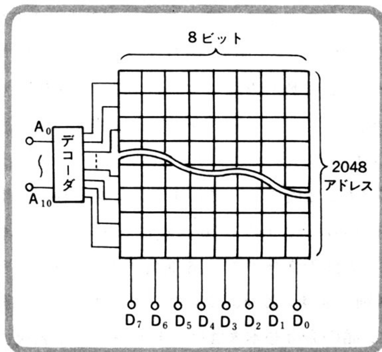
使い方による分類



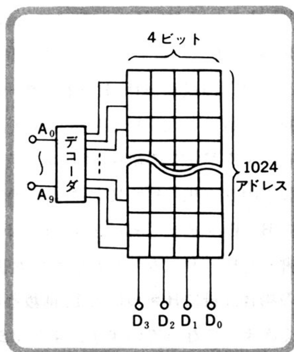
主に
ROMはプログラムを書いておく → **読み出しだけだが電源を切っても消えないから**
RAMは一時記憶用データ格納 → **読み書きできるから**

よく使われるメモリの例

EPROM (2716)



スタティック RAM (2114)



メモリは容量，生産工程，パッケージなどが各種あります．メーカのマニュアルによって適当なものをさがすことも重要な仕事です．



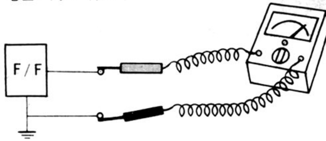
11 ビットパターンと16進表現

CPU内部や入出力信号線は，電気的に電圧が高い状態と低い状態の二つで意味を持つようになっています．1本の電線では信号が有る，無い，の二つの状態でしかあり得ませんが，複数の電線を一束にして考えると，高低の組み合せは2の累乗で増加します．アドレスバスは16本ありますが，16本の電線の高低の組み合せは$2^{16}=65536$通りになります．すなわち，CPUの出すアドレス情報は65536のメモリを識別できることになります．データバスは8本ですから，データとしては$2^{8}=256$通りの種類が得られます．しかし数回に分割してやりとりをすれば理論的には無限の組み合せが得られます．8本では数値としては0～255までしか表現できないはずですが，実際にはもっと広い範囲の数値を扱えるのはこのためです．

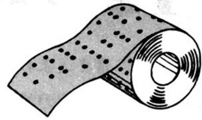
一つの2値状態を表現する単位を**ビット**と呼びます．アドレスバスは16ビットです．電圧の高い状態のことを"H"または"1"と呼び，低い状態のことを"L"または"0"と呼びます．16ビットの状態を表現するのに"1"と"0"を16個並べてもよいのですが，長くて扱いにくいので困ります．そこで"1"と"0"の組み合せ(**ビットパターン**)を2進数とみなして，これを16進数に変換して扱うと大変便利になります．要するに，ビットパターンを4桁ずつ区切って"1""0"の組み合せに名前をつけたと考えればわかりやすいでしょう．名前は数字の0～9，次が10でもよいのですが，2桁になってしまうのでAと呼び，次にB，C～Fまであります．0～Fで16ありますので16進数になるのです．本書でもビットパターンをいちいち書くのは大変なので，16進数で表現します．その場合は後にHをつけて10進数と区別します．Z-80のアセンブリ語でもこう書くきまりになっています．またマシン語の命令も本来はビットパターンで表現されるはずですが，一般的に16進数表現で扱います．

演算はビットパターンを2進数として行ないます．普通は0～255（8ビットでは）になりますが，符号を付けて−128～+127として考えることもできます．どちらをとるかでプログラムは異なります．

❶ 電圧が高い（低い）



❷ あながあいている（あいてない）



❸ スイッチが上に倒れている（下に倒れている）



❹ ランプが点燈している（消えている）



などを“1”というとき，（　）内のときは“0”という ➡ 誰かが決めた

※　4つ要素が並ぶとすると下のとおり16通りのパターン（組み合せ）が考えられる．

0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	A
1 0 1 1	B
1 1 0 0	C
1 1 0 1	D
1 1 1 0	E
1 1 1 1	F

いちいち書くのは大変なのでパターンに名前をつけることにした（右欄）．

要素が4つだから ――→ 16通り＝2^4
要素が8つだったら → $2^8=256$通り
要素が16だったら ――→ $2^{16}=65536$通り

※ パターンを2進数と見なすと名前は16進数になる．
コンピュータ内の演算はこのパターンを2進数と見なすことにより数値を表現して行なわれるようになっている．

※ たった8本の電線（信号）でも256の状態を表現することができる．

- 1のことを“H”（high），0のことを“L”（low）とよぶこともある．
- 普通は“0”になっていて，“1”になったら「信号がある」と決めておけば，“1”のことをアクティブという．
- 普通は“1”になっていて，“0”になったら「信号がある」と決めておけば，“0”のことをアクティブといい「この信号は負論理だ」という．

12 プログラムの実行

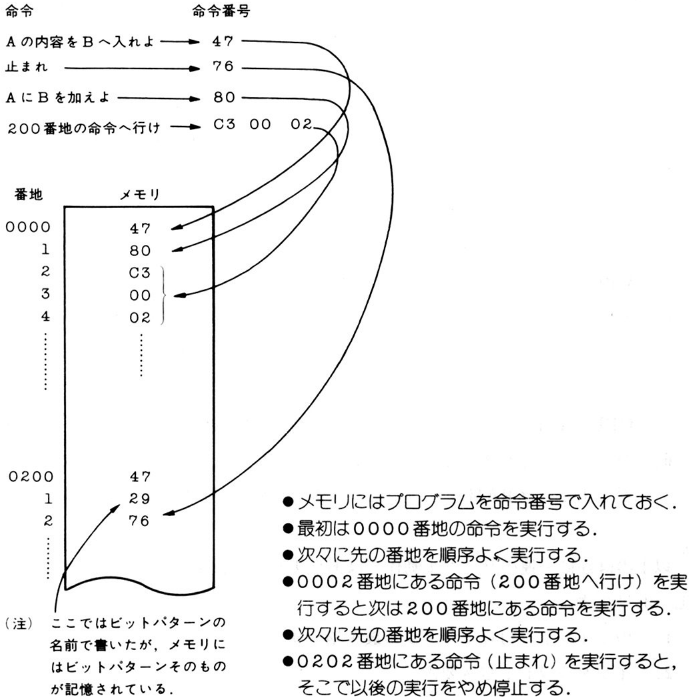
メモリに書き込まれているプログラムは，マシン語です．マシン語は英数字の組み合せで，人間の目にはきわめてわかりにくいものですが，たとえば「AとBを加えてAに入れよ」とか，「止まれ」とか，「Aの内容を10番のポートへ出力せよ」といった命令にそれぞれ付けられた番号なのです．CPUは，日本語や英語で書かれるより，番号のほうが簡単に見分けることができます．

プログラムの書き込まれたメモリには，これも番号＝番地（**アドレス**）がつけられています．実行するときは，まずゼロ番地に書かれた命令から順次1ずつくり上げていきます．ところが命令の中に「何番地へ飛べ」というのがあると，1ずつくり上げるのではなく，指定の番地へ飛び（ジャンプ）ます．また「前の演算の結果がゼロならば何番地へ飛べ」という命令では，条件によって，ゼロなら飛び，ゼロでなければ次の番地の命令を実行します．

サブルーチン コール命令があると，指定された番地へ飛び，そこから順次実行し最後につけられたリターン命令で，先程飛んできた番地の次へ戻ることができます．同じ手順を何度も使いたいときによく使う方法で，飛ぶ前のプログラムを**メインルーチン**，飛んでくるプログラムを**サブルーチン**と呼びます．

サブルーチンとよく似ているのが割り込みです．割り込みは，命令があって飛ぶのではなく，電気信号がCPUの端子に与えられると，どこを実行中であってもある番地へ飛び，リターン命令で元のメインルーチンへ戻ります．**割り込み処理ルーチン**と呼ばれます．

CPU内には，プログラムカウンタ（PC）と呼ばれるレジスタがあり，この内容で，いま実行すべき命令の入っているメモリのアドレスを示すようになっています．一つの命令を実行するたびにこのPCを増やし，次の命令のアドレスを指します．また，特定のアドレスへジャンプするようなときは，PCへ飛び先のアドレスを強制的に入れることにより目的を達します．



（注） ここではビットパターンの名前で書いたが，メモリにはビットパターンそのものが記憶されている．

- メモリにはプログラムを命令番号で入れておく．
- 最初は0000番地の命令を実行する．
- 次々に先の番地を順序よく実行する．
- 0002番地にある命令（200番地へ行け）を実行すると次は200番地にある命令を実行する．
- 次々に先の番地を順序よく実行する．
- 0202番地にある命令（止まれ）を実行すると，そこで以後の実行をやめ停止する．

13 CPUの信号のやりとり

CPUにはたくさんの信号入出力線が出ていますが，これらの信号線を使って基本的な六つの動作を時分割的に行ないます．それぞれを**マシンサイクル**と呼びます．

1. フェッチサイクル（M1サイクル）

   メモリに書き込まれているプログラム命令をCPU内の命令解析用レジスタ（インストラクションレジスタ）へ読み込み，解読する．

2. メモリリードサイクル

   メモリからデータをCPU内のレジスタへ読み込む．

3. メモリライトサイクル

   CPU内のレジスタからメモリへ書き出す．

4. IOリードサイクル

   入力ポートからデータをCPU内のレジスタへ読み込む．

5. IOライトサイクル

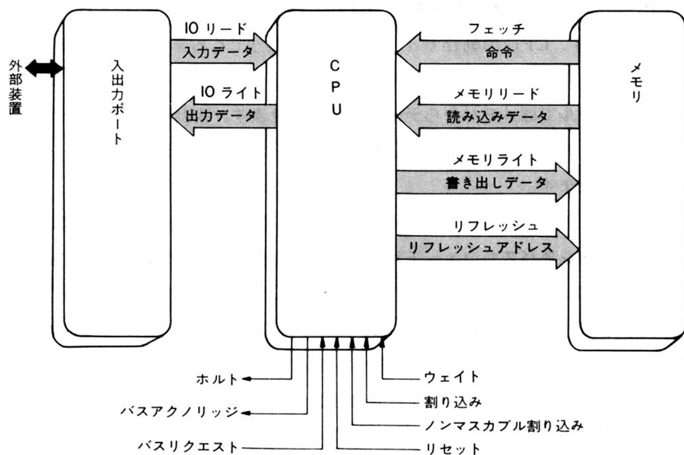
   CPU内のレジスタから出力ポートへデータを書き出す．

6. リフレッシュサイクル

   ダイナミックRAMをリフレッシュするためのアドレス信号を出す．

以上のほかに一連のサイクル動作ではありませんが，単独の意味を持つ信号が七つあります．

1. ウェイト；クロックサイクル数を増やす．
2. インタラプト；割込要求．
3. ノンマスカブルインタラプト；ノンマスカブル割込要求．
4. リセット；初期状態に戻す．
5. バスリクエスト；CPUの動作を停止させバスを空け渡す要求．
6. バスアクノリッジ；バスリクエストを受け付けた返事．
7. ホルト；CPUがホルト命令を実行し，停止状態に入ったことを外部へ知らせる．



は一連の動作（マシンサイクル）で，アドレス情報とデータの受け渡しを伴う．

は情報の受け渡しは伴わない．

14 データバス，アドレスバスとシステム制御信号

CPUにある8本の**データバス**は，CPUと外部とのデータを出し入れするための信号線です．CPUの動作サイクルによって乗ってくるデータの意味は異なります．フェッチサイクルでは，メモリから命令コードが乗ってきますし，メモリリードサイクルではメモリからデータが乗ってきます．またメモリライトサイクルとIOライトサイクルでは，CPU内のレジスタからのデータがデータバス上に乗せられます．

このようにデータバスは，時々によって種々の信号の出し入れに使われます．

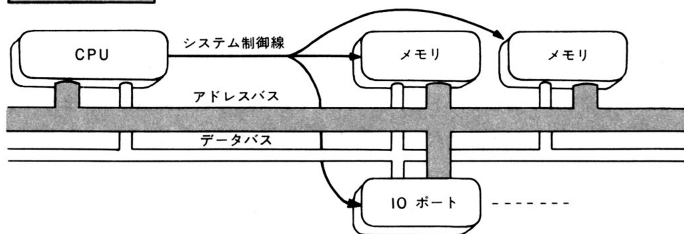
また，**アドレスバス**もフェッチサイクルでは実行すべき命令の入っているアドレスが出力され，メモリリード，ライトサイクルでは，読んだり書いたりするメモリのアドレスが出力され，また，IOライト，IOリードのサイクルでは，入出力すべきIOポートのアドレスが乗せられてきます．

データバスにせよアドレスバスにせよ目的の異なる信号が次々に乗せられますので，現在の信号は何を意味するかを識別するための別の信号線が必要になります．リード($\overline{RD}$)，ライト($\overline{WR}$)，メモリリクエスト($\overline{MREQ}$)，IOリクエスト($\overline{IORQ}$)，エムワン($\overline{M1}$)，リフレッシュ($\overline{RFSH}$)の各信号の組み合せが，この役割をしています．実際にはこれらの各信号がすべて同時に出たり止まったりするのではなく，それぞれのタイミングで変化します．アドレスバス，データバスの占有時間も目的により異なります．

CPUの信号のうちデータバスとアドレスバスだけは正論理すなわち，1のとき"H"レベル，0のとき"L"レベルとなります．他の信号線はすべて負論理で，普段信号のないとき"H"になっていて，必要なときだけ"L"になります．

負論理の信号名の略称には——(バー)を付けて表わします．

情報の受け渡し



システム制御信号線

- エムワン（$\overline{\text{M1}}$）
- メモリリクエスト（$\overline{\text{MREQ}}$）
- IO リクエスト（$\overline{\text{IORQ}}$）
- リード（$\overline{\text{RD}}$）
- ライト（$\overline{\text{WR}}$）
- リフレッシュ（$\overline{\text{RFSH}}$）

これらの組合せでアドレスバスに乗っている情報，データバスに乗っている情報あるいは，CPUがデータバスにどこから情報を乗せてもらいたいかを表現している．

※CPU は命令の内容によって，送り出すべき情報を送り出し，要求すべき情報を受けとる．
情報はデータバスを使う．
送り先や要求先は，システム制御信号とアドレスバスで指定する．

※メモリや IO ポートなど周辺回路は，上の CPU の要求に対しある時間内に正確に応答しなければならない．これは汎用 CPU の場合ユーザの利用技術にかかっている．

15 命令語の構成

マシン語の命令は，一つの機能を持つ命令が1～4バイトで構成されます．

1バイト命令は，オペコードだけで意味を持つ命令です．

2バイト命令は，オペコードと，オペランドに記述された数値を次の1バイトに持つものと，2バイトで一つのオペコードを意味するものがあります．

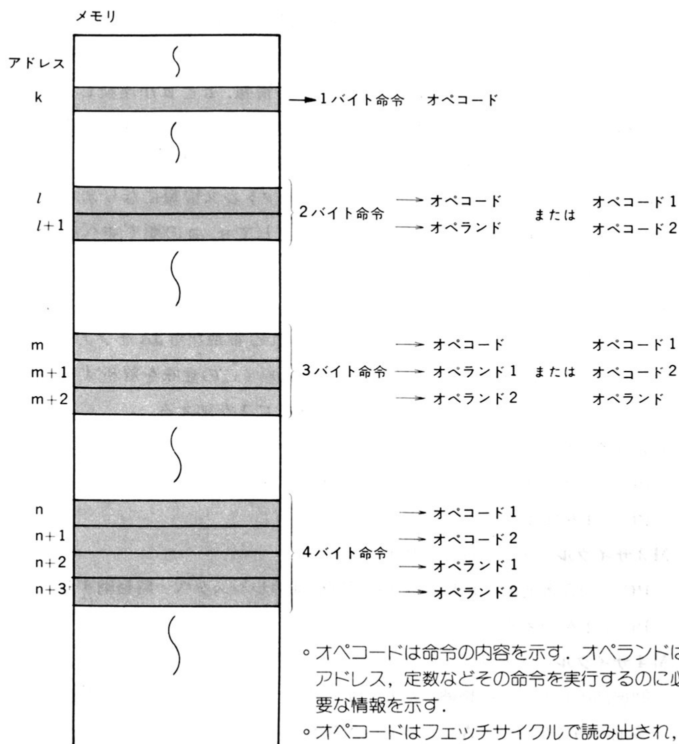
3バイト命令は，オペコードと，オペランドに記述された数値を次の2バイトに持つものと，2バイトのオペコードと1バイトのオペランドを持つものがあります．

4バイト命令は，2バイトのオペコードと2バイトのオペランド情報により構成されます．

オペコードが2バイトに渡る命令はZ-80特有のものです．最初の1バイトをフェッチし解読した時点で，2バイトのオペコードを持つことがわかりますから，あと1バイトをフェッチし解読します．したがってフェッチサイクルが二つあり，エムワン($\overline{M1}$)信号も2回出されます．

CPUはこの1～4バイトで構成される命令をフェッチサイクルとメモリリードサイクルを起動して全部読み込み，意味を解析し実行します．これが終ると次の命令へと順次読み込み，解析，実行をくり返していきます．

一つの命令を実行する一巡を，**命令サイクル**（インストラクション サイクル）と呼びます．



- オペコードは命令の内容を示す．オペランドはアドレス，定数などその命令を実行するのに必要な情報を示す．
- オペコードはフェッチサイクルで読み出され，オペランドはこれにつづくメモリリードサイクルで読み出される．

16 命令の実行

例として〔LD　A，(mn)〕という命令を分解してみましょう．

この命令には，3Aという番号がついていて，CPU内部では，この番号で意味がわかるようにできています．3Aの次に二つの情報，nとmが連続している3バイト構成の命令です．具体的には「mn番地のメモリの内容をCPU内のAレジスタへ入れよ」という意味があります．mとnは各8ビットで，メモリアドレスは16ビットで表わされますので，二つで一つのアドレス情報になります．このときメモリ上にはm（上位）とn（下位）を反対にしてn，mの順で並べる規定になっています．

M1サイクル（フェッチサイクル）

プログラムカウンタ（PC）の内容で示される番地から3AをフェッチするCPU内のインストラクションレジスタへ入れてこの意味を解析すると，次に何をすべきかわかり以下を実行する．PCに1を加える．

M2サイクル（メモリリードサイクル）

PCの内容番地よりnを読み込んで制御部のレジスタへ一時格納する．

PCに1を加える．

M3サイクル（メモリリードサイクル）

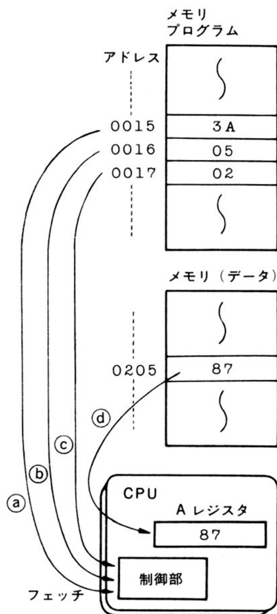
PCの内容番地よりmを読み込んで制御部のレジスタへ一時格納する．

PCに1を加える．

M4サイクル（メモリリードサイクル）

制御部のレジスタへ格納したnとmにより一つのアドレスmnを構成し，mn番地の内容を読んでAレジスタへ入れる．

以上の四つのマシンサイクルによって命令を実行したわけですが，命令の内容によって，それぞれのサイクルの組み合せは異なり，サイクル数も異なります．ただし，どんな命令でも必ず，フェッチサイクルが最初に起動され，フェッチした命令の内容によって次に何をするか決定します．



命令実行の一例

① 0015番地の命令を実行する順番がくると，CPUはアドレスバスに0015を乗せ，システム制御信号はメモリからの命令読み込みを要求する．
② メモリはこれに対し0015番地の中身をデータバスへ乗せる．
③ CPUはこれを受けとり命令の意味を解析する．

―― 以下解析の結果 ――

④ CPUはアドレスバスに0016を乗せ，システム制御信号によりメモリからの読み込みを要求する．
⑤ メモリはこれに対し0016番地の中身をデータバスに乗せる．
⑥ CPUはこれを受けとる．
⑦ 同様に0017番地の中身も受けとる．
⑧ いま読み込んだ0016と0017番地の中身をつないで0205という値を作り，これをアドレスバスに乗せ，システム制御信号によりメモリからの読み込みを要求する．
⑨ メモリはこれに対し0205番地の中身をデータバスに乗せる．
⑩ CPUはこれを受けとりAレジスタへ入れる．

（①～③がフェッチサイクルで各命令実行ごとにまずこのサイクルに入る．）

17 アセンブラ記法のルール −1−

16 進数で表現されるマシン語命令では，人間には覚えにくくわかりづらいため，直感に訴えるような英語の略語（ニモニック）を対応づけたのが**アセンブリ言語**です．アセンブリ言語で書かれたプログラムを**ソースプログラム**と呼びます．ソース プログラムをマシン語に自動的に置き換えるプログラムが**アセンブラ**です．このアセンブラ プログラムが読み込むソース プログラムの書き方には，一定のルールがあります．

ラベルはアドレスに付ける仮の名称です．この命令のアドレスへジャンプしたり参照したりするために必要なところだけでよく，つけなくてもよいのです．英文字で始まる 6 文字以内の英数字列で後で見てわかりやすい名前を付けます．

オペコードは命令語のニモニックです．CPU の持つ命令の中から必要なものを選んで使います．

オペランドは，オペコードによってはないものもあります．一つだけのものも，“，”で区切って二つ必要なものもあります．二つ必要な命令の場合は，前に書くのがディスティネーション，後に書くのをソースといって，ソースからディスティネーションへのデータの移動を意味します．ソースまたはディスティネーションをカッコでくくるときは，カッコ内のアドレスのメモリに入っている内容を意味します．カッコ内がレジスタ名のときは，そのレジスタに入っている内容をアドレスとしたメモリを意味します．

コメントはプログラムをあとで見たときにわかりやすくするために必要なことをメモしておく所です．的確なコメントの豊富につけられたプログラムは，誰が見てもよくわかり，大変便利なものです．

アセンブリ言語で書く命令には，上のようにマシン語命令に変換されるもののほかに，疑似命令といって，マシン語に変換されずに，アセンブラ プログラムに対して指示を与えるだけの命令もあります．また，アセンブラ プログラムがマクロ アセンブラと呼ばれる場合は，一命令をあらかじめ別に定義された数個以上のマシン語命令群に変換するマクロ命令といったものも使うことができます．

ラベル
ニモニック
ディスティネーション ｝オペランド
ソース
この定数が16進表現であることを明示する.

```
LOOP LD   A,B            Bレジスタの内容をAレジスタへ入れよ
     ADD  A,B            AレジスタへBレジスタの内容を加えよ
     LD   HL,0802H       HLレジスタへ定数0802Hを入れよ
     LD   HL,(0802H)     HLレジスタへ0802番地のメモリの内容を入れよ
     OUT  (00H),A        00H番地のIOポートへAレジスタの内容を出力せよ
     IN   A,(00H)        Aレジスタへ00H番地のIOポートから読み込め
     JP   LOOP           LOOP番地へ飛び，その命令以下を実行せよ
JOB1 LD   (HL),A         HLレジスタの内容をアドレスとするメモリへ
                         Aレジスタの内容を書き出せ
     END                 プログラムの終了をアセンブラプログラムへ知らせる
```

オペランドの表記法

	ディスティネーション	ソース
レジスタ名	レジスタ	レジスタ
定数	――――	定数
(レジスタ名)	レジスタの内容アドレスのメモリ	レジスタの内容アドレスのメモリ
(定数)	定数アドレスのメモリ	定数アドレスのメモリ

16進表現で，最上位がA～Fのアルファベットになるときは頭に0を付け，ラベル名と区別する.

(例)　A000Hは　0A000Hと書く

18 アセンブラ記法のルール －2－

行の先頭，すなわち前の行の最後につけられたC/R（キャリッジリターン）の次に続く文字列は，**ラベル**です．英大文字に続く英数字の組み合わせで，6文字以上は無視されます．ラベルの文字列の直後に“：”コロンがあれば，行頭でなくてもラベルとみなされます．ラベルをつけない行は，1個以上のスペースをラベルの代わりに入れておきます．

ラベルと1個以上のスペースで区切られた後にオペコードを書きます．命令表の中から選んで下さい．全部で74あります．

オペコードと1個以上のスペースで区切られた後には，そのオペコードに対するオペランドが続きます．二つ必要なときは“，”コンマで区切ります．このオペランドは，レジスタ名，16進表現の定数，10進表現の定数，フラグの状態（分岐の条件）名，を書きます．定数は，他の命令に付けたラベルを書いてもよく，ジャンプする場合などは特に絶対番地を計算しなくてもよいので便利です．オペランドのソースやディスティネーションに（　　）カッコを付けたときは，その内容を番地とするメモリの内容を意味します．

オペランドに書く定数は，10進表現で書いてもよいし，16進表現で書いてもよいのですが，16進表現で書くときは，後にH（ヘキサデシマルの頭文字）をつけておきます．また，先頭がアルファベットA～Fで始まるときはさらに前に0をつけてラベルでないことを特徴づける約束になっています．

コメントは行中のどこにでも“；”セミコロンがあれば，それ以後は自由にコメントエリアとして使うことができます．ただし1行は通常80字位までで切られることがありますので，長くなる場合は数行に分けます．

Z-80のアセンブリ言語は標準的な取り決めがありますが，アセンブラ プログラムの種類（メーカや機種）により若干の違いがあります．アセンブラを使うときは，それぞれの説明書を一読して下さい．

```
        Z80 MACRO ASSEMBLER  V1.1  PAGE    1

 1                      ;IPL
 2                              ORG*    0FF00H
 3 FF00 3EED                    LD      A,0EDH  ;MODE
 4 FF02 D3F3                    OUT     (0F3H),A        ;MDR0
 5 FF04 3E37                    LD      A,037H  ;SLP
 6 FF06 D3F2                    OUT     (0F2H),A        ;CDR
 7 FF08 DBF6            LP0:    IN      A,(0F6H)        ;ISR
 8 FF0A CB7F                    BIT     7,A     ;CCE
 9 FF0C 20FA                    JR      NZ,LP0  ;CCE WAIT
10 FF0E 0610                    LD      B,16    ;BLOCK COUNT
11 FF10 210000                  LD      HL,0000H        ;ADDRESS
12 FF13 3EFF            LP1:    LD      A,0FFH  ;WDC SET
13 FF15 D3F1                    OUT     (0F1H),A        ;WDC
14 FF17 3E3B                    LD      A,03BH  ;RDL
15 FF19 D3F2                    OUT     (0F2H),A        ;CDR
16 FF1B DBF6            LP2:    IN      A,(0F6H)        ;ISR
17 FF1D CB6F                    BIT     5,A     ;DA
18 FF1F 2007                    JR      NZ,LP3
19 FF21 DBF0                    IN      A,(0F0H)        ;DBR
20 FF23 2F                      CPL
21 FF24 77                      LD      (HL),A  ;DATA LOAD
22 FF25 23                      INC     HL      ;ADDRESS COUNT UP
23 FF26 18F3                    JR      LP2
24 FF28 CB7F            LP3:    BIT     7,A     ;CCE
25 FF2A 20EF                    JR      NZ,LP2  ;CCE/DA  WAIT
26 FF2C 10E5                    DJNZ    LP1     ;END?
27 FF2E C300E0                  JP      EXIT    ;RETURN TO MONITOR
28                      ;
29      E000            EXIT:   EQU*    0E000H
30                              END*
```

コメント

定数（番地）の代りにラベルで指定

コメント行（見やすくするために一行あける）

行番号

アドレス

マシン語命令（オブジェクト）

ラベル

オペコード

オペランド

ソースプログラム

アセンブラが自動的に変換する

タイプライタキーを打って入力する

* ORG：以下のプログラムを配置するアドレスを指示する．
EQU：ラベル名の文字列をオペランドの数値とすることを指示する．
END：プログラムの終りを明示する．

19 命令の分類

機能別に命令を分類すると，次のようになります．

8 ビットロード

1 バイトの情報をレジスタ間，レジスタとメモリ間で転送する．1 バイト定数をレジスタかメモリへ入れる．

16 ビットロード

2 バイトの情報をレジスタ間，レジスタとメモリ間で転送する．2 バイト定数をレジスタへ入れる．

レジスタ交換

レジスタの内容を入れ替える．

メモリブロック転送

メモリ内の複数バイトのブロックを別のアドレスへ転送する．

メモリブロックサーチ

メモリ内ブロックに指定の情報があるかどうか探す．

8 ビット演算，論理演算

1 バイト単位の加減算，論理和，論理積，排他的論理和，カウントアップ，カウントダウン，比較をする．1 または 2 の補数をとる．

16 ビット演算

2 バイト単位の加減算，カウントアップ，カウントダウンをする．

ローテートシフト

レジスタ，メモリのビットパターンを回転させまたは左右へずらす．

10 進補正

2 進化 10 進数として扱うデータの加減算後の補正をする．

フラグ操作

キャリフラグを "1" にする．反転させる．

CPU 制御

プログラムの実行と，割り込みを制御する．

ビット操作

レジスタ，メモリの特定の1ビットを"0"か"1"にする．また，判定する．

ジャンプ

プログラムの実行を条件によりまたは無条件で，指定のアドレスへ移行させる．

コール，リターン，リスタート

プログラムの実行を条件によりまたは無条件で，指定のアドレスへ移行させる．ただしこのとき，リターン命令によりもとのアドレスへ戻れるような手順を含んでいる．

入力出力

指定のIOポートへレジスタとの間で1バイト情報を入・出力する．

連続入出力

メモリブロックを1バイト単位に連続して出力する．

メモリブロックへ1バイト単位に連続して入力する．

マイナス数の表現（2の補数）

2進数でマイナスを表現するには，多くの場合次の手法がとられます．

4ビットだけで考えれば，ゼロすなわち0000のひとつ前の−1は1111になります．なんとなればこれに1をたすと桁上りをして10000ですが，4ビットだけで考えているので桁上りを無視せざるを得ないからです．

左端ビットが"1"のときをマイナス数と決める．

0	0000	8	1000	→	−8
1	0001	9	1001	→	−7
2	0010	A(10)	1010	→	−6
3	0011	B(11)	1011	→	−5
4	0100	C(12)	1100	→	−4
5	0101	D(13)	1101	→	−3
6	0110	E(14)	1110	→	−2
7	0111	F(15)	1111	→	−1

上記のとおり，4ビットでは0〜F(10)または−8〜+7を表現することができます．正数を負数に変えるには，ビットの"1""0"を反転させて1を加えればよいのです．

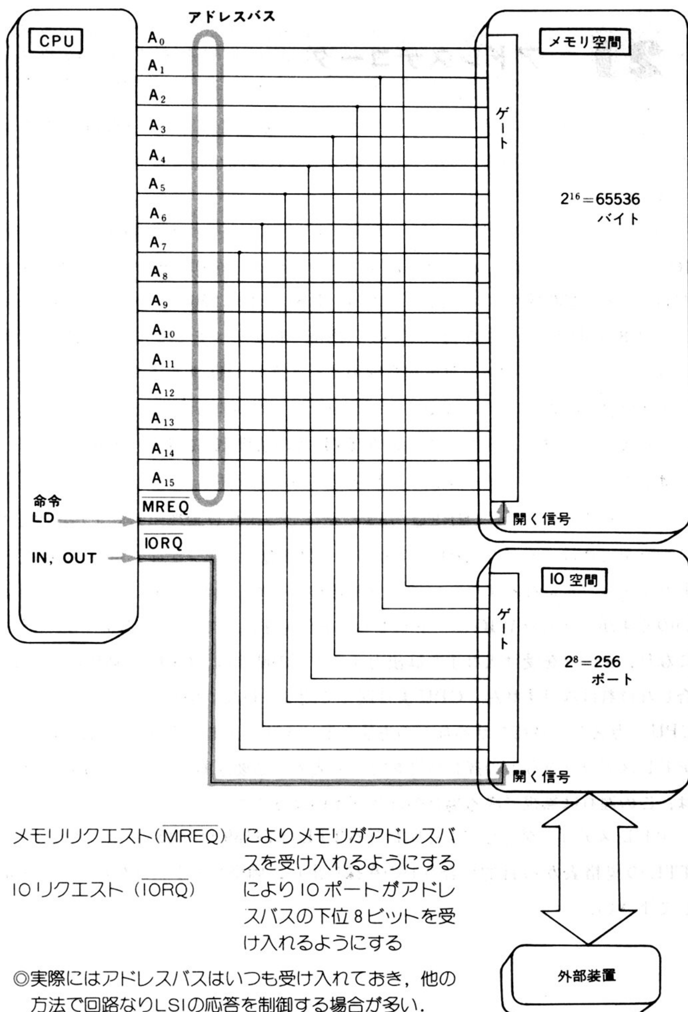
20 メモリ空間とIO空間

Z-80では（他の8ビットCPUもほとんど）接続できるメモリは最大64Kバイト（Kは1024）です．これはアドレスバスが16本あるので $2^{16}=64$K となるからにほかなりません．メモリとは別にIOの領域として256のアドレスがあります．したがってIOポートは256まで接続できることになります．これは，インプット，アウトプットのサイクルではアドレスバスの下位8ビットのみ使用しているためで，$2^{8}=256$ ということです．同じアドレスバスをメモリリクエスト信号とIOリクエスト信号により切り替えて使用するためにメモリ空間とIO空間を別々に持ったことになります．

メモリ空間へのアクセスは，ロード命令を使い，IO空間へのアクセスはイン，アウト命令を使います．IO空間へのアクセスのほうが，タイミング的に長い時間がとれるようになっています．

17項の説明では省略しましたが，IO空間のアドレスを示す場合もOUT（OOH），A のようにカッコに入れるきまりになっています．

バイト
- o8ビットを1バイトと呼ぶ．8ビットのCPUはバイトマシンと呼ばれ，8ビットを同時に扱うので，メモリは1アドレスのアクセスで8ビット（データバス8本）の読み書きが行なえる構成とする．
- oしたがって，1アドレス4ビットのメモリ（2114など）では2個，1アドレス1ビットのメモリ（4116など）では8個並列に並べる．



メモリリクエスト($\overline{MREQ}$)　によりメモリがアドレスバスを受け入れるようにする

IOリクエスト（$\overline{IORQ}$）　によりIOポートがアドレスバスの下位8ビットを受け入れるようにする

◎実際にはアドレスバスはいつも受け入れておき，他の方法で回路なりLSIの応答を制御する場合が多い.

21 アドレスデコーダ

アドレスバスに出力されるビットパターンは16本の端子から65536通りもありますから，どのパターンのときはどのメモリを選ぶかを決めてやらなければなりません．

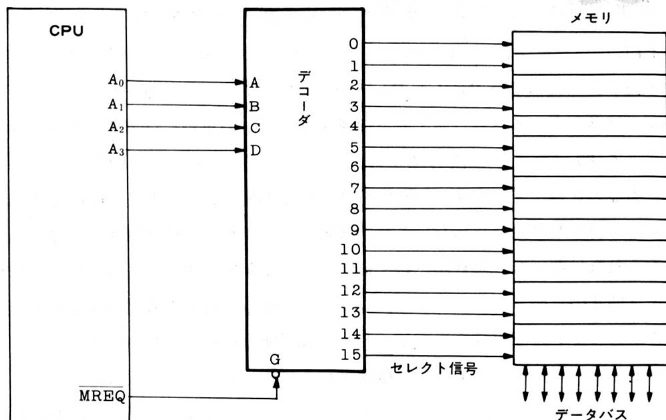
アドレスバスに出てくるアドレス信号のビットパターンから特定のメモリやIOポートを選び出すセレクト信号(イネーブル信号)を作り出すわけです．メモリはたとえば256バイトのメモリであれば，LSIの中に256バイトすなわちアドレスバス8本分のデコーダを持っていて，内部で特定のメモリセルを選択してくれます．ところが，このメモリを複数使用するときや，IOポート用のペリフェラルを複数使用するときは，CPUとペリフェラルやメモリの間にアドレスデコーダが必要になります．デコーダは通常TTL等の論理ICで構成します．

番地だけでなく，メモリ空間とIO空間の切り替えも，ここで行なうのが普通です．メモリリクエスト($\overline{\mathrm{MREQ}}$)がアクティブつまり負論理出力ですから"L"になったときはメモリが，IOリクエスト($\overline{\mathrm{IORQ}}$)がアクティブならばIOポートがイネーブルになるようなロジックを組むのです．通常問題になることは少ないのですが，メモリやIOポートへセレクト信号を出してから実際にイネーブルになり，データを受け入れまたは出力するまでの時間は，CPUの動作時間に適合しなければなりません．CPUより遅くて間に合わない場合はウェイト信号をCPUに与えて一時待たせるなどの方法をとります．メモリやIOの応答時間と，アドレスデコーダなどの遅れ時間を含めて考える必要があります．しかし一般には，この遅れは無視できる場合のほうが多いようです．

アドレスデコーダとしてよく使われるTTLは74139や74155などがあります．TTLの規格表から目的に合うものを探し出し，価格や入手状況を検討して決定して下さい．

（例） 16バイトメモリのアドレスデコーダ

16バイト＝2^4：4本のアドレスバスで16のアドレスを表現できる



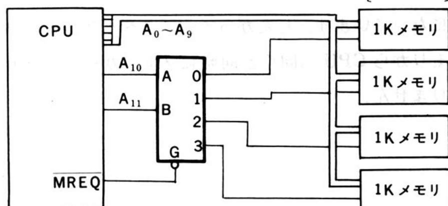
デコーダの出力

（X は何でもよい）

G	D	C	B	A	出力
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	2
0	0	0	1	1	3
0	0	1	0	0	4
0	0	1	0	1	5
0	0	1	1	0	6
0	0	1	1	1	7
0	1	0	0	0	8
0	1	0	0	1	9
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	×	×	×	×	ナシ

- $\overline{MREQ}$ がノンアクティブ（負論理だから“1”）のときはメモリセレクトはしない．
- $\overline{MREQ}$ がアクティブ（“0”）のときだけ A_0 から A_3 のビットパターン16種により16本の出力のどれかへセレクト信号を出す．
- セレクトされたメモリはデータバスとデータの受け渡しを行なう．

（参考） 1Kバイトのメモリ4個を使ったときのデコーダ $\left[\begin{array}{l}1K=1024\\ \quad=2^{10}\end{array}\right]$



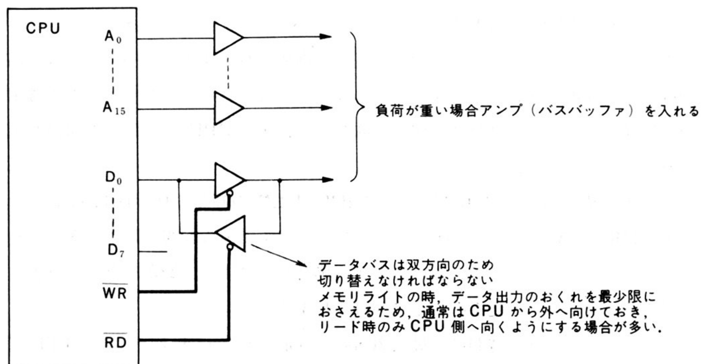
22 バスバッファ

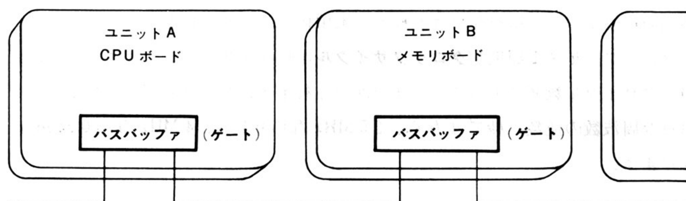
TTL のロジック IC には，ファンイン，ファンアウトという概念があります．マイコンの周辺の場合 "H" と "L" のほかに電気的に切り離されたハイインピーダンス状態をとるスリーステートの端子が多いため単純には考えられなくなります．Z-80 ファミリの出力端子は，一つのノーマルタイプの TTL をドライブできる能力を持っています．LS タイプであれば四つまでです．これ以上の素子をドライブしなければならないときや接続ケーブルを長く引き回すときはバッファを入れなければなりません．特に長いケーブルを使用するときは波形の乱れやノイズ防止のため，バスドライバ，レシーバの能力に特別の考慮が望まれます．ときにはフォトカップラやシュミットトリガが使われることもあります．

メモリなどが何枚ものボードで構成される場合，ボードの出入口のところにバスバッファを設けるのが普通です．単にバッファだけでなく，ゲートの働きもして，そのボード内のアドレスがアクセスされたときだけゲートが開くようにしておけば，CPU に対して無駄な負荷をかけることもなく，デバッグもやりやすくなります．

データバスのバッファは双方向性です．リード，ライトの信号によって方向を決定します．マルチ CPU システムとしたときや DMA を使用したときはきわめて複雑になりますので，システムのブロック化をするときには十分考慮する必要があります．

割り込みを使用するシステムでは，ペリフェラルは，CPU がメモリからフェッチしてくる命令を横からデータバスを見ていて，割り込みからの復帰を知るようになっています．したがって，データバスのバッファはフェッチサイクルではメモリから CPU へ向くと同時にメモリからペリフェラルへも送り込まなければなりません．





◦ ゲートはユニットの機能により，開け閉めする場合もあるが，常時開けておき他のゲートを制御するための情報を受けとる場合もある．

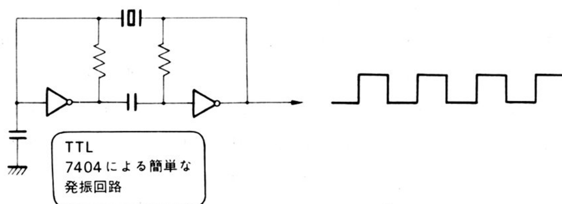
23 クロック

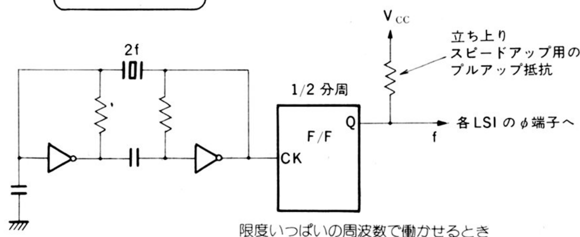
Z-80 ファミリは，ノーマルバージョンのLSIは，最高2.5 MHz，A バージョンと呼ばれるLSIは，4 MHz までのクロックを与えることができます．システムのすべての動作はこのクロックに同期して進められますので，当然周波数が高いほど処理速度は速いわけです．しかし周辺に接続されるメモリや入出力の端末の応答速度が追従しなければ，何の意味もなく，単にCPUを待たせる回路が増えるだけになります．

クロック(Φ)は単相の方形波ですが，"H"と"L"の幅がそれぞれ180 ns(ノーマル) 110 ns (A) 以上となっています．したがって周波数が規定以下であってもデューティ比が1になっていないとまずい場合があります．2倍の発振を分周すれば正確にデューティ比1のパルスが得られます．

立ち上り立ち下りの時間は，最大30 ns と規定されています．これはTTLでは実現できない値ですが，かなりのオーバースペックのようです．実際にはTTLの出力にプルアップ抵抗を付ける程度で実用化している例があります．

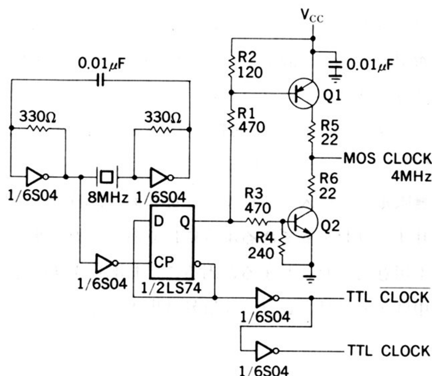
クロックパルス1周期を**クロックサイクル**と呼びます．マシンサイクルは，クロックサイクル数個によって一つずつ進められます．1クロックサイクルは，(クロック周波数の逆数) 秒ですから，2.5 MHz なら0.4 μs，4 MHz なら0.25 μs となります．





限度いっぱいの周波数で働かせるとき

〔理想的な回路例〕（ザイログ社マニアルより）



24 リセット

CPUのリセット($\overline{RESET}$)端子へのリセット信号は，パワーオン時には与えなければなりません．CPU以外へも同じ信号を与えます．PIOとDMAはパワーオン リセットの機能を内部に持っていますので必要ありませんが，与えてもよいのです．リセットの時間は，電源電圧が安定し，クロックが安定に与えられてから最低3クロック サイクル分（2.5 MHz クロックであれば，1.2 μs）以上です．実用上問題なければ，長いほど安全といえます．

CPUは，このリセット信号により

プログラム カウンタ(PC)

I レジスタ

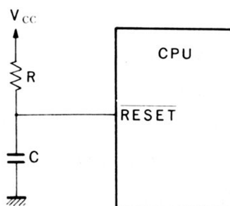
R レジスタ

をゼロにして，割り込みモードを0として，割り込みを受け付けない（インタラプト ディスエーブル）状態にします．これ以外のレジスタ（もちろん，CPU外のRAMも）は変化しません．

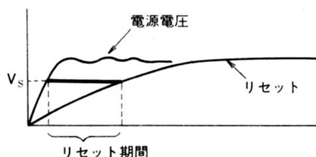
リセット信号がノンアクティブ（負論理ですから"H"）になると，まず，ゼロ番地の命令をフェッチすることから動作を始めます．もしIレジスタを使用するなら，ゼロ番地からのプログラムでIレジスタへ必要な数値を書き，割り込みモードを希望のモードに設定し，その他必要な初期化を行なってから，割り込みを受け付け可の状態（インタラプト イネーブル）にする〔EI〕命令を実行させます．またスタックポインタ(SP)の設定もしなければなりません．

リセット信号は，パワーオン時に与えることはもちろんですが，プログラムが意図しない無限ループに入ってしまうこともあり，特にデバッグ中には，手動で電源を入切せずにリセットできるようにしておくと大変便利です．ただし何らかの動作中に不用意にリセットすると困ることもありますので注意が必要です．リセット期間中はメモリリフレッシュ信号は出ません．

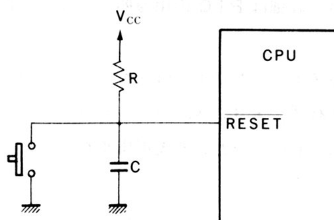
パワーオンリセット



RとCにより数 ms の時定数を持たせる



マニアルリセット



パワーオンリセットも兼ねる．スイッチのチャタリングは
Cにより吸収される．

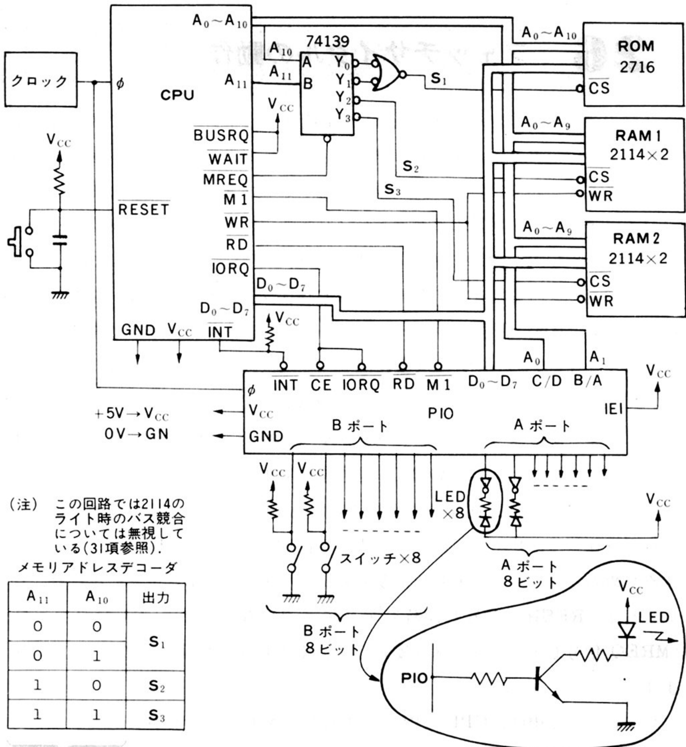
* リセットにノイズがのることを想定すると，シュミットトリガ回路を通した方が確実といえます．

25 システム構成

マイコンシステムの構成は，目的によって大きく異なります．普通はCPU 1個に対して，入出力のポート数あるいはビット数を満たすだけのIOペリフェラル，そしてプログラムを収容しきれるだけのROMと，データエリア（ワーキングエリア）としてのRAM，IOとメモリのアドレスデコーダ，クロック，リセットで構成されます．さらに大きなシステムではDMAを採用したりCTCを追加したり，場合によってはCPUを複数にしたマルチCPUシステムとすることもあります．マルチCPUシステムは，目的の仕事がはっきり区分できるときはプログラムが楽になり，効率がよくなる点でメリットがあります．

入力にスイッチ8個，出力にLED 8個，プログラムは2Kバイト以下，データエリアは2Kバイト以下，といった簡易なコントローラの例を図に示します．メモリは合計4Kバイトですので，アドレスバスは下位12ビットだけを使い，上位4ビットはあそばせてあります．したがって16進数で表現すると3桁だけが有効で上位桁は何であってもよいわけです．何であってもよいということは，何であってもいずれかのメモリがセレクトされてしまうので，これ以上増設することはできなくなります．IOポートにはPIOを1個使ってありますので，アドレスは四つだけ必要（詳細はＰＩＯの項参照）になり，下位2ビットだけ有効です．

このシステムでもスイッチをマイクロスイッチや光センサに，LEDをリレーやソレノイドに変えれば，メカニズムのコントロールロジックを置き換えたり，簡単なシーケンスコントローラとして実用化することができます．



メモリアドレスデコーダ

A_{11}	A_{10}	出力
0	0	S_1
0	1	
1	0	S_2
1	1	S_3

メモリアドレス配置

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	16進表現
ROM	×	×	×	×	0	0	0	0	0	0	0	0	0	0	0	0	×000
	×	×	×	×	0	1	1	1	1	1	1	1	1	1	1	1	×7FF
RAM 1	×	×	×	×	1	0	0	0	0	0	0	0	0	0	0	0	×800
	×	×	×	×	1	0	1	1	1	1	1	1	1	1	1	1	×BFF
RAM 2	×	×	×	×	1	1	0	0	0	0	0	0	0	0	0	0	×C00
	×	×	×	×	1	1	1	1	1	1	1	1	1	1	1	1	×FFF

各メモリLSIの中でデコードしてくれるので
最小値と最大値のみを記入した

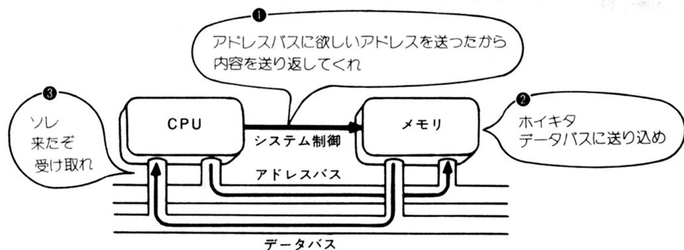
26 フェッチサイクルの動作

一つの命令を実行するとき，最初にCPUはフェッチサイクルに入ります．プログラムカウンタの値に示されるアドレスのメモリからオペコードを読み込み，解析します．そしてプログラムカウンタへ1を加え次の読み込みにそなえます．

フェッチサイクルが始まるとアドレスバスへプログラムカウンタの値が乗せられ，同時にエムワン($\overline{\text{M1}}$)信号が"L"すなわちアクティブになります．アドレスバス上のデータの安定するのを待ってメモリリクエスト（$\overline{\text{MREQ}}$）信号が，続いてリード（$\overline{\text{RD}}$）信号がアクティブになってメモリをアクセスします．3番目のクロックT_3の立ち上りで，CPUはデータバスの内容を読み込み（サンプリング）ます．T_2が立ち下るときにウェイト（$\overline{\text{WAIT}}$）信号がアクティブになっていると，そのままの状態でウェイトサイクルT_WがT_2とT_3の間に入ります．T_Wの立ち下りでもウェイト（$\overline{\text{WAIT}}$）信号を見ますので次へ進めたいときは，ノンアクティブ"H"に戻さなければなりません．応答の遅いメモリを使うときに，これで時間待ちさせることができます．T_3ではダイナミックRAMをリフレッシュするためのリフレッシュアドレスがアドレスバスの下位7ビットに出され，リフレッシュ（$\overline{\text{RFSH}}$）信号も同時にアクティブになります．メモリリクエスト($\overline{\text{MREQ}}$)信号もアドレスバスが安定しているであろう期間にアクティブになります．

リフレッシュ中は，CPUの内部では命令の解析が行なわれ，次のサイクルに何をすべきか判断しています．8ビットの汎用レジスタ間の転送命令などでは4クロックのフェッチサイクルだけで実行を完了してしまいます．

フェッチサイクルのメモリリクエスト（$\overline{\text{MREQ}}$）信号は，データバスをサンプリングする約1.5クロック前に出されますので，この時間内にメモリが応答しなければなりません．しかし問題になることは少なく，万一のときもウェイト（$\overline{\text{WAIT}}$）信号で解決できます．



クロックサイクル
マシンサイクル
フェッチサイクル
リフレッシュサイクル
T_1　T_2　T_3　T_4　4クロックサイクル必要

クロック
アドレスバス　プログラム・カウンタ　Rレジスタ
$\overline{\text{MREQ}}$
$\overline{\text{RD}}$
$\overline{\text{M1}}$
$\overline{\text{RFSH}}$
データバス

ここで$\overline{\text{WAIT}}$を調べ"L"になっているとT_2と同じ状態が1クロック(T_w)続く．T_wの立ち下り時にも同様に調べる．

メモリからのデータ
ここで読み込む
メモリ応答に必要な時間

システム制御信号

- フェッチサイクル
  $\overline{\text{MREQ}}$と
  $\overline{\text{RD}}$と
  $\overline{\text{M1}}$がアクティブ
- リフレッシュサイクル
  $\overline{\text{MREQ}}$と
  $\overline{\text{RFSH}}$がアクティブ

27 メモリ リード サイクル

メモリからデータを読み出して，レジスタや他のメモリへ転送する命令の1サイクルとして，メモリ リード サイクルへ入る場合と，命令語のオペランドを読み出すためのメモリ リード サイクルがあります．動作はどちらも同じで，通常[*1] 3クロック サイクルの間に実行されます．

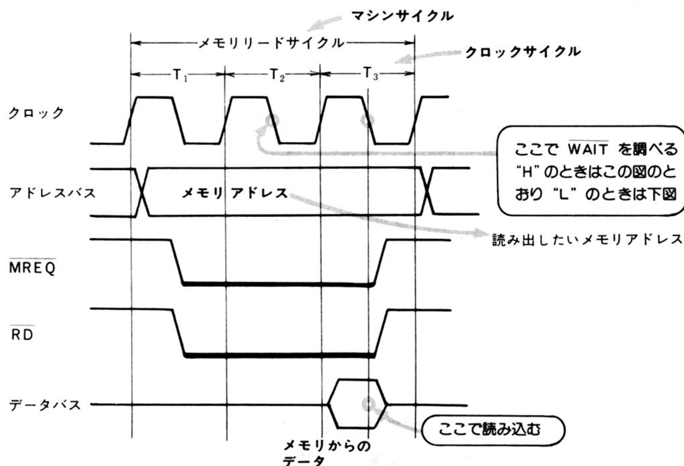
アドレス バスに読み込むべきアドレスが送出され，これが安定するのを待ってメモリ リクエスト（$\overline{\text{MREQ}}$）信号とリード（$\overline{\text{RD}}$）信号が同時に出されます．2番目のクロックは，メモリが応答し，データ バスへ情報を乗せてくるのを待つ時間です．もしメモリの応答が間に合わない場合には，立ち下りの前後にウェイト信号を入れることができます．3番目のクロックの立ち下り時点に，CPUはデータ バスのゲートを開き，内部へデータを読み込みます．

リード（$\overline{\text{RD}}$）とメモリ リクエスト（$\overline{\text{MREQ}}$）信号が出てからCPUが読み込むまでの時間は，フェッチ サイクルでは1.5クロックでしたが，メモリ リード サイクルでは2クロックあります．クロックを2.5 MHzとすると，1クロック当りの時間は400 ns[*2] ですから，フェッチのときは600 ns，リードのときは800 nsとなります．したがって，アクセス タイムが450 nsぐらいの標準的なメモリであれば問題なく使用できます．

---

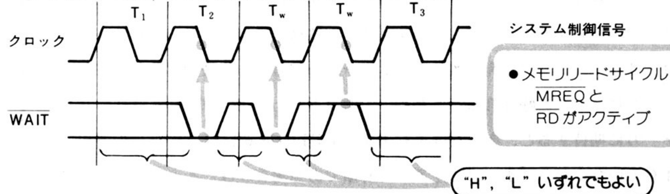
*1 命令により例外はある．

*2 ns：ナノセックといい．10^{-9}秒．



ウェイトサイクル (T_w) を含む場合

(フェッチサイクル, メモリライトサイクル, IO サイクルも同じ)



T_w では信号は T_2 と同じ状態を保っている

28 メモリライトサイクル

CPU からメモリに対する書き出しのマシン サイクルです．レジスタからメモリ，メモリからメモリへの転送命令などで実行されますが，サブ ルーチン コールのコール命令で，戻り番地をスタッカへ格納するときや，割り込みがかかったときも実行されます．いずれも同じタイミングで通常* 3クロックです．前項のリード サイクルとほとんど同じで，異なるのはリード（$\overline{RD}$）信号が出ずに，ライト信号が出る点と，CPU からデータバスへデータが乗せられる点です．

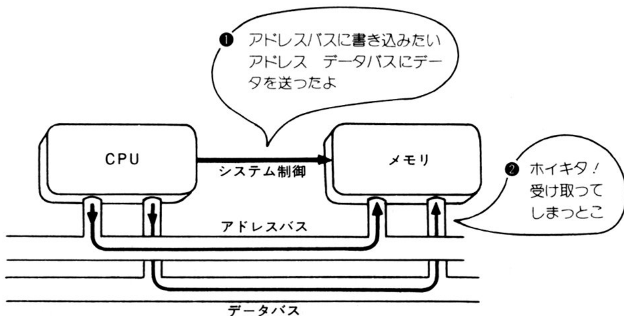
ライト（$\overline{WR}$）信号は，メモリ リクエスト（$\overline{MREQ}$）信号より1クロック遅れて出されます．これは，データ バスが安定するのを待つためで，ライト（$\overline{WR}$）信号でメモリの読み込みのゲートをコントロールすることができるようになっています．

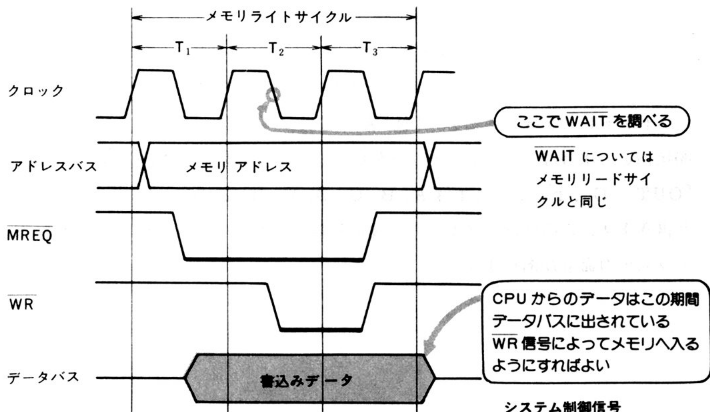
メーカの立場とユーザの立場

マイクロプロセッサは，一つの電子部品との見方からすると，その使い方は，ユーザの責任において研究されなければなりません．真空管やトランジスタでも，メーカの発表するのは一使用例にすぎず，回路構成や応用製品の信頼性については，すべてユーザの研究や雑誌等のレポートを参考にして使用されてきました．

コンピュータと呼ばれることから，大型コンピュータ並の指導やサービスをメーカに期待すると，あてがはずれます．LSI の単価には，これらの費用は含まれていないからです．マニュアルも有償である場合がほとんどです．マニュアルは，指導書ではありませんので，解りにくいものです．基本的に「何をどうするとどうなります」という LSI の機能を述べ連らねてあるにすぎません．書物や雑誌に目を向け，いろいろな事例に接することが大切です．

* 命令により例外はある．





システム制御信号

メモリライトサイクル
$\overline{\text{MREQ}}$ と
$\overline{\text{WR}}$ がアクティブ

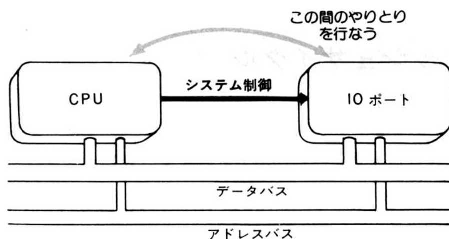
29 IOリードサイクル，IOライトサイクル

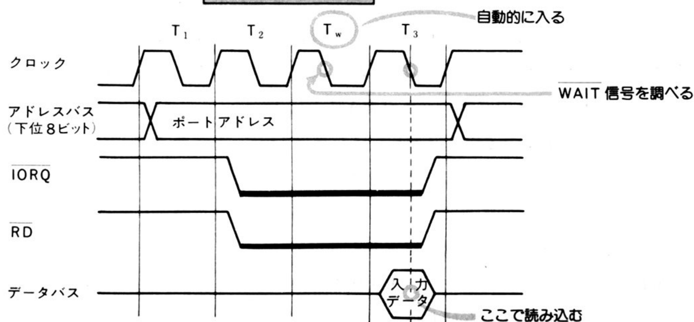
IOポートに対するリード，ライトは，アドレスバスの下位8ビットが有効なアドレスとして使われます．上位8ビットへは何らかの状態が出力されますので，無視しなければなりません．

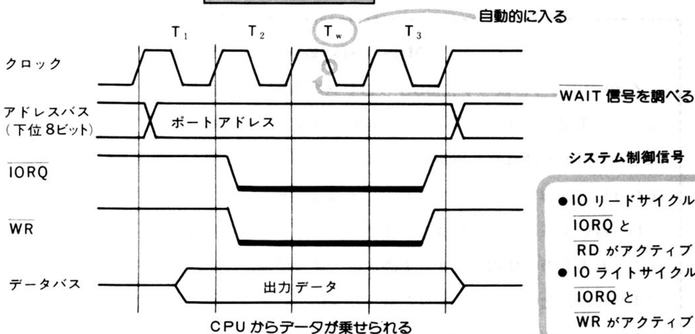
イン〔IN〕命令，アウト〔OUT〕命令と，これらの繰り返しの命令を実行したときに，このサイクルへ入ります．

IOサイクルでは，メモリの読み書きより多く時間を必要とすることが想定されますので，ウェイトサイクルT_Wが自動的に入ります．もっと長く時間待ちさせたいときは，このT_Wの立ち下りの前後にウェイト（$\overline{WAIT}$）信号を入れることになります．またリード（$\overline{RD}$），ライト（$\overline{WR}$）とIOリクエスト（$\overline{IORQ}$）各信号は，立ち下り，立ち上り共同じタイミングです．

イン〔IN〕命令，アウト〔OUT〕命令は，入出力するポートアドレスを指定しなければなりません．直接指定の場合はAレジスタとの間でのやりとりになり，ポートアドレスをnとすると〔IN　A,（n）〕，〔OUT　（n），A〕と書きます．間接指定の場合は，ポートアドレスをCレジスタに入れておき，〔IN　r,（C）〕，〔OUT　（C），r〕（ここでrはA，B，C，D，E，H，Lのいずれかのレジスタ）と書きます．このほかにメモリの一連の複数バイトのブロックを入出力するブロック入出力命令があります．







システム制御信号

- IO リードサイクル
  IORQ と
  RD がアクティブ
- IO ライトサイクル
  IORQ と
  WR がアクティブ

30 リフレッシュサイクル

リフレッシュサイクルは，各フェッチサイクルの後半に行なわれます．アドレスバスの下位7ビットにはRレジスタの内容が出力されます．Rレジスタは1回ごとに1ずつ増やされ，ダイナミックRAMの1列ごとに再書き込みを行ないます．

このときはリード($\overline{RD}$)，ライト($\overline{WR}$)の信号の代わりにリフレッシュ($\overline{RFSH}$)信号が出て，リフレッシュサイクルであることを知らせます．

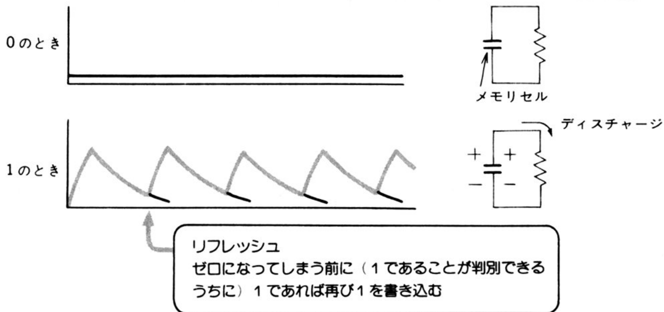
リフレッシュは長くても2ms以内に1回以上行なわなければならないので，Z-80を最高速で働かせたときでも16Kビットのダイナミック RAMが限界になります．16Kビットのダイナミック RAMの代表的なものに4116と呼ばれるものがありますが，$16K = 2^{14}$で，アドレスバスは14ビット必要です．ところが上位，下位7ビットずつを時分割して与えますので，アドレス入力は7端子しかありません．リフレッシュの場合は，7ビットのアドレスだけを順次与えればよいのです．したがってRレジスタは7ビットとなっています．

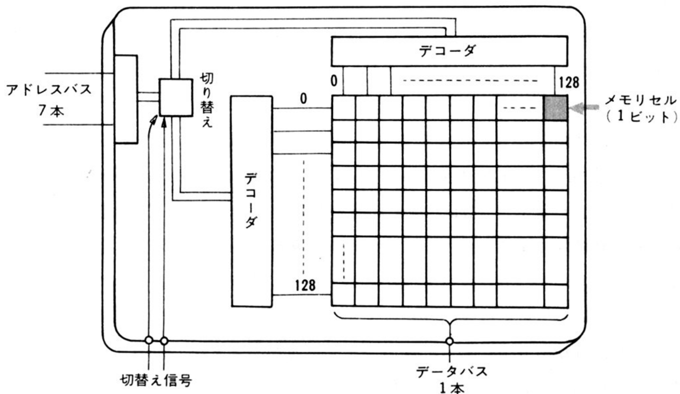
16Kビット以上のダイナミックRAMのリフレッシュは，Z-80の機能とは別にリフレッシュ回路を作らなければなりません．

MOS LSIの取扱い

Z-80ファミリに限らず，MOSプロセスのLSIは，端子のインピーダンスが高いため帯電しやすく，チップの絶縁が破壊することがあります．保存するときはモスパッドと呼ばれる導電性のスポンジにさしてピン間をショートしておけばまちがいがありません．また，ピンに直接手で触れたり，帯電していそうな所へさわらないようにしなければなりません．ハンダゴテも絶縁のよいものか，アースのとれるものを使用します．実際には，そう簡単にこわれてしまうわけではありませんが，念のために心がけたほうがよいでしょう．

ダイナミックRAMの模式図（4116）





- アドレスバス7本へ　上位7ビット，下位7ビットを時分割で与える．
- いま，上位が与えられているか下位が与えられているかを切替え信号で知らせてやる．
- リフレッシュの場合は，一方だけを与え，他方は全メモリセルをセレクトする．→ データ出力は無意味（デタラメ）→ データ出力なし．

31 CPUと周辺の接続（インターフェース）

各マシンサイクルごとにアドレスバスとデータバスの意味が変わりますので，どの動作をしているかによって，メモリとIOポートのどれを選ぶか，また，書き込みか読み出しかの切り替えをしなくてはなりません．

まず，アドレスバスをデコードして，セレクト信号を作ります．次にメモリリクエスト（$\overline{\text{MREQ}}$）とIOリクエスト（$\overline{\text{IORQ}}$）信号によりメモリとIOを切り替えます．ここでIOリクエスト信号はエムワン（$\overline{\text{M1}}$）信号と同時にアクティブになったときは別の意味（割り込み応答）になりますので，このときはIOをセレクトしてはいけません．Z-80ファミリではセレクトしても問題ありません．

IOポートにZ-80ファミリのLSIを使うときは，同じ名称のピン同士を接続すれば，システム制御信号に関しては何ら心配はいりません．2個以上のLSIを接続するならば，アドレスデコードしたチップセレクト信号を使わなければなりません．

メモリは種類によって異なります．ROMの場合は，チップセレクト信号は，リード信号が出たときだけ与えるようにしなければなりません．RAMの中にはアウトプットドライブ（$\overline{\text{OD}}$）端子があるものがありますが，この端子へはリード信号を直接与えます．ライトイネーブル（$\overline{\text{WE}}$）端子へはライト信号を与えればよいのです．アウトプットドライブ端子がないRAMの場合，ライトイネーブル端子へ与える信号には工夫がいります．メモリライトサイクルでは，メモリリクエスト信号が出て少したってからライト信号が出ますが，このライト信号が出るまでの時間，メモリはリード動作をしてしまい，CPUからのデータとメモリからのデータがデータバス上でぶつかってしまいます．実際問題としてはLSIをこわしてしまうほどのことはないようですが，さけたほうが賢明でしょう．
リード信号とリフレッシュ信号がなくかつ，メモリリクエスト信号が出たときだけライトイネーブル端子へ与える信号を作れば解決できます．

システム制御信号

	$\overline{M1}$	$\overline{MREQ}$	$\overline{IORQ}$	$\overline{RD}$	$\overline{WR}$	$\overline{RFSH}$	
フェッチサイクル	○	○		○			メモリ
メモリ　リード		○		○			
メモリ　ライト		○			○		
IO　リード			○	○			IOポート
IO　ライト			○		○		
リフレッシュ		○				○	ダイナミックメモリ
割り込み応答	○		○				IO　ポート
		メモリ（アドレス空間）	IOポート（アドレス空間）	データの方向	データの方向		

◦この信号の組み合せでメモリやIOポートをイネーブル（活かす）したり，ディスエーブル（殺す）したりする（ように接続する）．
アドレスデコーダをコントロールしてセレクト信号が出るようにしたり，止めたりしてもよい．

◦Z-80ファミリで構成するときは，同じ信号線どうしを接続すればよい．

32 ウェイト信号とホルト信号

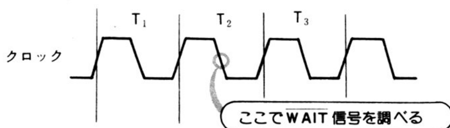
ウェイト（$\overline{WAIT}$）**信号**は，メモリやIOの応答速度がCPUのタイミングに間に合わないときに，ダミーのクロックサイクルを入れて，CPUを待たせるときに与えます．CPUは各マシンサイクルの2番目のクロックの立ち下りの時点で，ウェイト信号端子の状態を検知して，このとき"L"になっていれば，次のクロックをウェイトサイクルとして何もしないで待ちます．このウェイトサイクルの立ち下りのときも同じく検知していますので，"H"になるまで空転していることになります．ウェイト中はリフレッシュは行なわれません．

IOライトサイクルとIOリードサイクルでは一つ，割り込み応答のフェッチサイクルでは二つのウェイトサイクルが自動的に挿入されます．

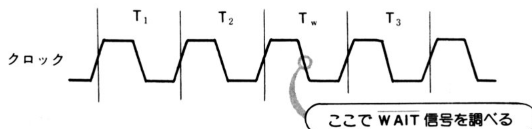
簡単なシステムではウェイトの端子を使用しないことのほうが多いと考えられますが，そのときはV_{CC}へ続いて"H"レベルにしておかないと，CPUは動作したり止まったり，不安定な状態になってしまいます．

ホルト（$\overline{HALT}$）**信号**は，CPUがホルト〔HALT〕命令を実行したときに出される信号です．〔HALT〕命令を実行すると，プログラムカウンタの進行を止めてしまいますので，CPUが停止したのと同じです．普通のプログラムでは，このようなことは起こり得ませんので，異常があったことを外部へ知らせる信号として使えます．ホルト状態からの解除は，リセット信号を与えるか，割り込み受け付け可になっているときは，割り込みをかければよいのです．ホルト状態では，ノンオペレーション〔NOP〕命令といって，フェッチサイクルだけで何もしない命令をくり返し実行しています．このようにしてある理由は，ホルト中もメモリリフレッシュを絶やさないためです．

T_w がない場合

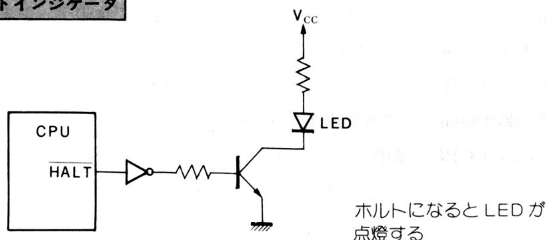


T_w がある場合



- T_2 の立ち下りまたは T_w の立ち下りで $\overline{WAIT}$ を調べ，"H" ならそのまま，"L" なら T_w を挿入する．
- 挿入された T_w の立ち下りでも $\overline{WAIT}$ を調べ，"H" なら次は T_3，"L" ならもうひとつ T_w を挿入する．
- $\overline{WAIT}$ が "H" になるまで何回でも T_w が入る．

ホルトインジケータ



33 割り込みの概念

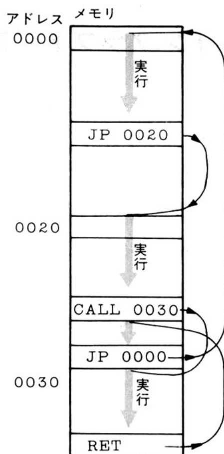
CPUは，リセット信号($\overline{\mathrm{RESET}}$)が入ると，ゼロ番地から順次命令を実行します．命令の中に，「ある番地へジャンプせよ」という命令，ジャンプ〔JP〕命令やコール〔CALL〕命令があると，その番地の命令を実行します．つまり，通常はプログラム カウンタを一つずつ繰上げていき，特別の命令があると，プログラム カウンタの内容をその命令に従って変更します．

いま，このようにして処理を進めているときに，急に他の処理をしなければならなくなったことを考えてみます．停電になってとりあえずバッテリに切り替えるとか，端末装置から故障を知らせる信号が入ったときに，通常のプログラムの進行を止めて，対策処理をしなければならないことがあります．このようなときに，インタラプト($\overline{\mathrm{INT}}$)かノンマスカブル インタラプト信号($\overline{\mathrm{NMI}}$)を入れることによって，プログラム上の命令とは関係なく，特定の番地へジャンプさせることができます．これを**割り込み**と呼びます．

割り込み機能をもっと積極的に使うことにより，処理効率を高めたり，プログラムを簡単にしたりすることもあります．たとえば，CPUからプリンタにデータを打出すよう命令を与えてからプリンタが動作を終るまでの時間は，CPUの処理速度に対して非常に長いのですが，この間ただ待つのではなく，CPUは別の仕事をしていて，プリンタからの動作終了割り込みで次のデータを打出すことができます．一度に二つの仕事をしているようにみえるのでマルチ タスクとかマルチジョブと呼んで，入出力の多い仕事ではCPUの処理時間は無視され，端末装置の動作時間だけで処理速度がほぼ決まってきます．

割り込みはCPUの動作に同期せずに，外部からのハード的な要求で呼び出されるサブ ルーチン コールと考えてよいでしょう．

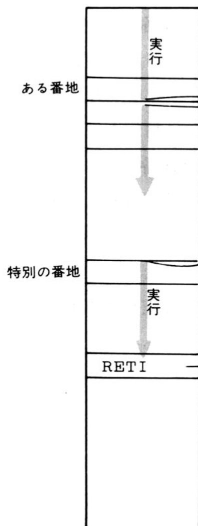
通常の実行



- 普通は順序よく．
- ジャンプ，コールなどの命令があれば指定の番地へ実行を移し，そこから実行する．

命令により実行アドレスを変える

割り込み



電気信号（$\overline{NMI}$ か $\overline{INT}$ 端子）割り込み信号

- ある番地の命令を実行中（命令は何でもよい）電気信号が入ると特別の番地へ実行を移す．RETI（リターン命令）があると元の命令の次（さっきのつづき）へ実行をもどす．

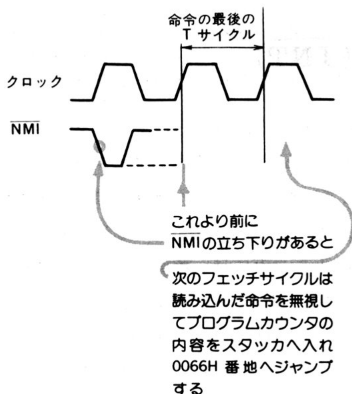
割り込み処理ルーチン

電気信号により実行アドレスを変える

34 ノンマスカブルインタラプト($\overline{NMI}$)

Z-80には，ノンマスカブル インタラプト（$\overline{\text{NMI}}$）と，ただのインタラプト（$\overline{\text{INT}}$）の2系統の割り込みがあります．ノンマスカブル インタラプトは，いかなるときでも受け付ける最優先の割り込みです（ただし，バス アクノリッジ中は受け付けません）．したがって，停電などシステムやオペレータにダメージを与えるような非常時の対策に使うことが多いようです．

ノンマスカブル インタラプトは名前のとおり，プログラムによってマスク，すなわち無効にすることができない割り込みです．CPUはノンマスカブル インタラプト信号線（$\overline{\text{NMI}}$）が立ち下ったとき，実行中の命令が終り次第，割り込み処理に入ります．割り込みの処理プログラムが終ったときまた元の番地へ戻らなければならないために，現在のプログラム カウンタ（PC）の内容をスタッカへ格納します．次に，プログラム カウンタへ0066Hを書き込み，新しいフェッチ サイクルへ入ります．したがって，0066H番地をコールすることになります．割り込みがかかった時点で実行中の命令は終りまで実行し，次の命令のフェッチ サイクルもあたかも平常のとおりに実行されますが，データバスを無視して〔CALL 0066H〕をフェッチしたかのように動作を続けていると考えられます（ただし，タイミングは異なります）．0066H番地には割り込み処理プログラムを書いておかなければなりません．そして処理の最後にリターン フロム ノンマスカブル インタラプト〔RETN〕命令があれば，スタッカへ格納しておいた戻り番地をとり出して，割り込みがかかったときに実行していた次の命令へ復帰します．（$\overline{\text{NMI}}$）が入ってから〔RETN〕が実行されるまではインタラプト（$\overline{\text{INT}}$）は無視され，待たされることになります．



$\overline{\mathrm{NMI}}$
がかかると

次のフェッチサイクル
(ダミー)

メモリ
ライト
サイクル

メモリ
ライト
サイクル

フェッチ
サイクル

5クロック

PCはこの命令
のアドレスを指
している

PCの上位
8ビットを
スタッカへ
書き込む

PCの下位
8ビットを
スタッガへ
書き込む

PCに0066H
を入れ0066H
番地の内容を
とり出し解析
する

割り込み
処理ルーチン

0066H番地からの
割り込み処理ルーチ
ンが終ったときに,
次に実行すべき続き
へ戻るため, PCに
入っている戻り番地
をスタッカへしまっ
ておく

フェッチ
サイクル

フェッチ
サイクル

メモリリード
サイクル

メモリリード
サイクル

フェッチサイクル

RETN : 1バイト目

RETN : 2バイト目

リターン・フロム・ノンマスカブル
インタラプト命令があると

スタッカより戻り
番地をPCに入れ

先程のダミー
にしてしまっ
た番地の命令
を再度フェッ
チし実行を続
行する

35 インタラプト（INT）

リセット信号が入ってCPUが初期状態から実行し始めると，インタラプト信号に対するマスクは，ディスエーブルすなわち割り込みを無視するようになっています．また割り込みモードは0になっています．もしモード1か2で使用するのなら，〔IM1〕か〔IM2〕の命令を実行させておく必要があります．モード2の場合は，Iレジスタとペリフェラル内のレジスタに対し数値を設定しなければなりません．これらの準備が終ったところで，割り込みを許可する命令，イネーブル インタラプト〔EI〕を実行させますと，これ以降割り込みがかかるようになります．割り込みを禁止するときは，ディスエーブル インタラプト〔DI〕です．

割り込みがかかると，次の割り込みがかからないようにディスエーブル インタラプト〔DI〕命令を自動的に実行し，おのおのの番地へジャンプします．割り込み処理プログラムの終りには，リターン〔RET〕命令が書かれていれば，CPUは元の処理を続行するべく戻り番地へジャンプします．ただしこのとき，〔RET〕命令の直前に〔EI〕命令を置かなければ，割り込みは禁止されたままになってしまいます．〔EI〕はただちに有効になるのではなく，次の1命令を実行し終ったときから有効になりますので，〔RET〕が終了した後に割り込みを受け付けるようになります．

なお，ノンマスカブル インタラプトは信号の立ち下りエッジでかかりますが，インタラプトのほうは各命令の最後のクロック サイクルの立ち上り時点で信号の状態を検査しますので，このとき"L"になっていなければなりません．

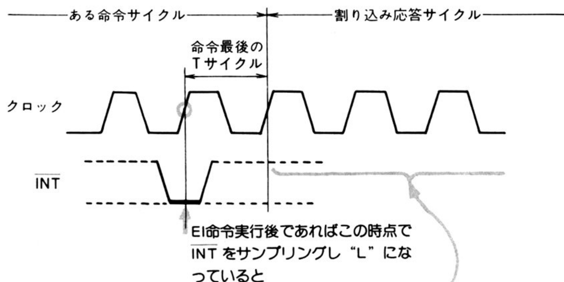
Z-80ファミリのペリフェラルは，優先順位を決定する機能を個々に持っており，外部に回路を必要としません．この機能を使うときは，割り込み処理が終ったことを通常のサブ ルーチンからのリターンとは区別してペリフェラルに与えなければならないため，〔RET〕ではなくリターン フロム インタラプト〔RETI〕命令を使用します．

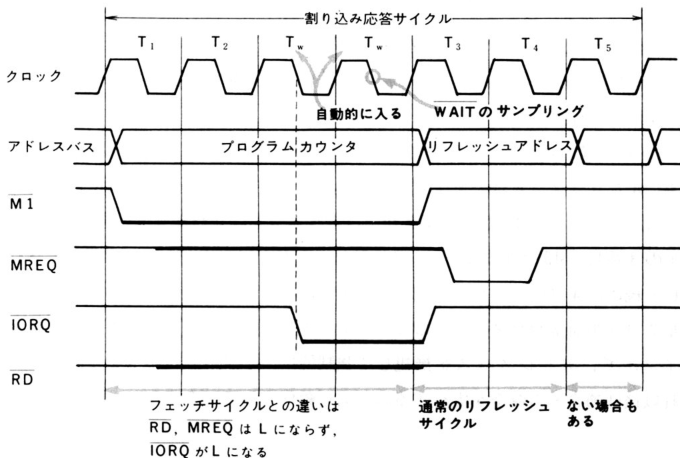
IM 0 ⎫
IM 1 ⎬ 割り込みモードを設定する命令．いずれかのモードを選びプログラムの最初の方で１回実行させる．
IM 2 ⎭

EI　　割り込み信号を許可する命令

DI　　CPU 側で割り込み信号を受け付けなくする命令

EIを実行後 CPU は割り込みを受け付け，DIを実行すると受け付けなくなる．

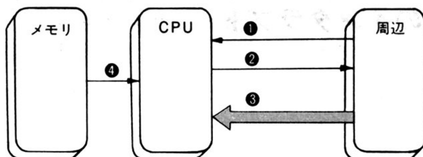




36 モード0のインタラプト

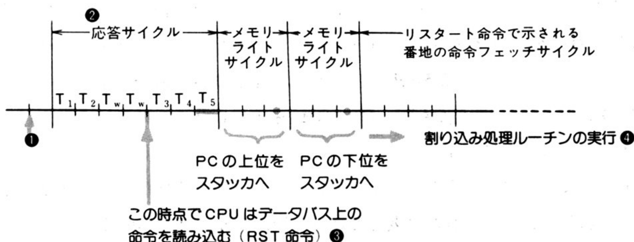
割り込みモード0は，8080Aと同じ動作をします．命令の最後のクロック サイクルの立ち上りのとき，インタラプト（$\overline{\text{INT}}$）信号が“L”になっていると，次のフェッチ サイクルは割り込み応答サイクルとなります．フェッチ サイクルとの違いは，メモリ リクエスト（$\overline{\text{MREQ}}$）信号が出ずにIOリクエスト（$\overline{\text{IORQ}}$）信号が出力されることと，3，4番目のクロック サイクルにウェイト サイクルが2個，自動的に挿入されていることです．したがって，ペリフェラル側では，エムワン（$\overline{\text{M1}}$）信号とIOリクエスト（$\overline{\text{IORQ}}$）信号が共にアクティブ，すなわち"L"になったことで，割り込みがかかったことを知ります．割り込み応答サイクルでは，アドレスバスには次の命令のアドレス，すなわちプログラム カウンタの値が出ますが，外部からはこれとは関係なく，何らかの命令をデータ バスに乗せてやります．この命令は主にリスタート〔RST〕命令か，コール〔CALL〕命令を使います．〔RST〕命令は1バイト構成ですからこれでよいのですが，〔CALL〕命令のときは3バイト構成ですので，あと2回メモリ リード サイクルに合わせて，データ バスへ割り込み原因に応じたプログラム ルーチンを実行させるための命令の続き（コール命令のジャンプ先番地を意味するオペランド）を与えなければなりません．

CPUの読み込みタイミングは，フェッチサイクルと同じくT_3の立ち上りです．またリスタート命令を与えたときは，スタックポインタの変更をするためのクロックサイクルT_5が付きますが，コール命令ではこれがなく，マシンサイクルの3番目（M3サイクル）が，メモリリードサイクルの3クロックの後にT_4として続き，4クロックサイクルとなります．これは通常の場合にフェッチサイクルでコール命令やリスタート命令をフェッチしたのと同じです．すなわち応答サイクルで，アドレスバスを無視して強制的にデータバスへ命令を与えれば，それ以後は通常その命令を実行するのと同じ動作をしています．

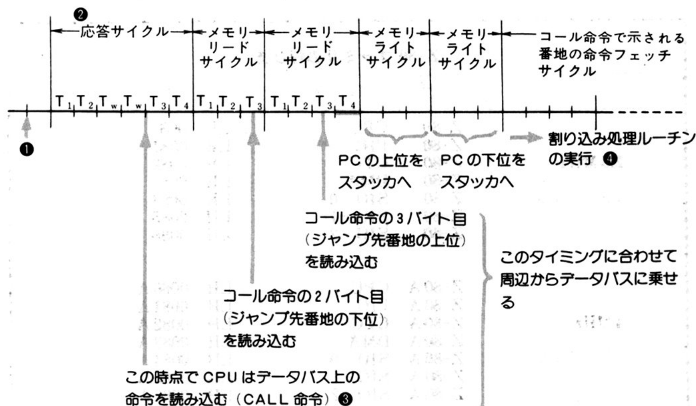


❶ 割り込み信号を与える
❷ 受け付けた返事（応答サイクル）
❸ データバスを通じて命令コードを与える
❹ CPUは❸で受け取った命令を実行する

リスタート〔RST〕命令を与えた場合



コール〔CALL〕命令を与えた場合



37 モード1のインタラプト

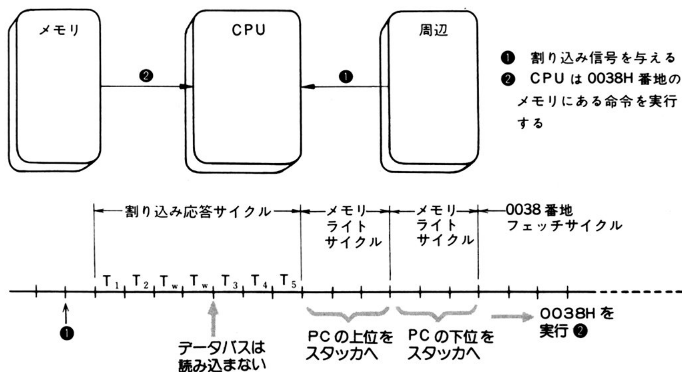
割り込みモード1は最も簡単な割り込みで，ほとんどノンマスカブル インタラプトと同じ動作をします．

インタラプト（$\overline{\text{INT}}$）信号の検知と，応答サイクルはモード0と同じです．応答サイクルでは，外部からは何も与えず，CPUも何も読み込まないので，ダミー サイクルとなります．次に現在のプログラム カウンタの値，すなわち戻り番地をスタッカへ格納するためにメモリ ライト サイクルが2回続き，そのあとプログラム カウンタを0038Hにします．次はこの番地の命令実行サイクルへ入ります．見かけ上は，〔CALL 0038H〕を実行したことになります（ただし，タイミングは異なります）．

このモードでは，割り込み要因によって，ジャンプ先の処理ルーチンを分けることはできません．もしこれが必要なときは，0038H番地からのルーチンの中で，外部からのステータスを読み込み，解析しなければなりません．

Z-80 ファミリの呼び方

	ザ イ ロ グ	シ ャ ー プ
2.5 MHz バージョン	Z-80 CPU Z-80 PIO Z-80 CTC Z-80 DMA Z-80 SIO-0 Z-80 SIO-1 Z-80 SIO-2	LH 0080 LH 0081 LH 0082 LH 0083 LH 0084 LH 0085 LH 0086
4 MHz バージョン	Z-80 A CPU Z-80 A PIO Z-80 A CTC Z-80 A DMA Z-80 A SIO-0 Z-80 A SIO-1 Z-80 A SIO-2	LH 0080 A LH 0081 A LH 0082 A LH 0083 A LH 0084 A LH 0085 A LH 0086 A


メモリ
CPU
周辺
❷
❶
❶ 割り込み信号を与える
❷ CPUは0038H番地のメモリにある命令を実行する
割り込み応答サイクル
メモリライトサイクル
メモリライトサイクル
0038番地フェッチサイクル
T_1 T_2 T_w T_w T_3 T_4 T_5
❶
データバスは読み込まない
PCの上位をスタッカへ
PCの下位をスタッカへ
0038Hを実行❷

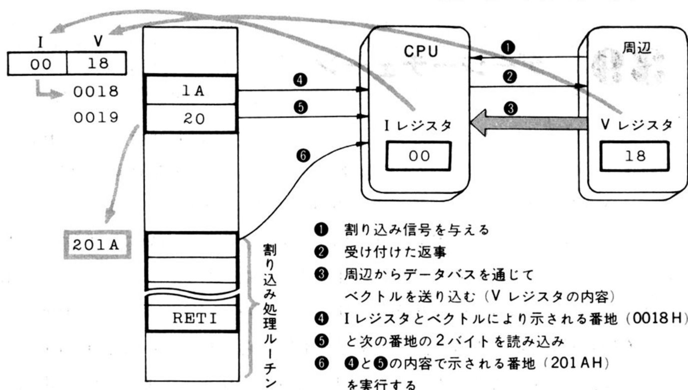
38 モード2のインタラプト

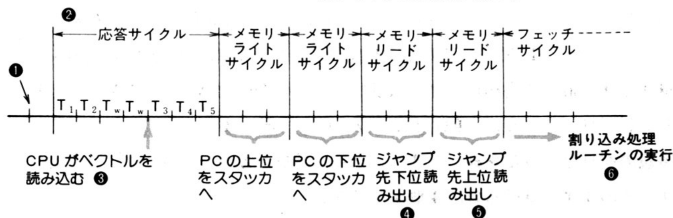
割り込みがかけられた次のフェッチサイクルは，割り込み応答サイクルとなります．他のモードと違うのは，エムワン（$\overline{M1}$）信号とIOリクエスト（$\overline{IORQ}$）信号が共に"L"になったら，ペリフェラルからはベクトルと呼ばれる1バイトの値をデータ バスに乗せ，CPUがこれを受け取ることです．次に戻り番地として現在のプログラム カウンタの値をスタッカへ格納するためのメモリ ライト サイクルが2回続きます．その次のサイクルでは，CPUは2バイトの情報をメモリから読み込むためのメモリ リード サイクルが2回続きます．そしていま読み込んだ値をプログラム カウンタへ入れて，そのアドレスからの実行を開始します．

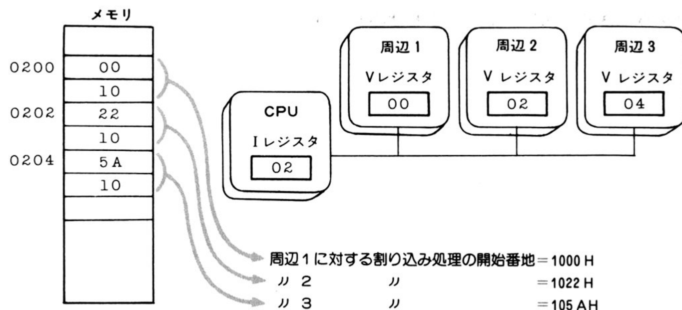
ベクトルとは，各ペリフェラルにあらかじめ書き込んである1バイトのデータで，これがアドレスの下位8ビットとなります．上位8ビットは，CPU内のIレジスタへ，これもあらかじめ書き込んでおきます．割り込みがかかると，Iレジスタの内容とベクトルで構成される番地と，次の番地のメモリの内容を2バイト読み込んで，この値の番地へジャンプすることになります．

ベクトルの値は，ペリフェラル個々に異なった値を書いておけば，割り込みをかけたペリフェラルによって別々の処理ルーチンへジャンプしますし，DMAやSIOでは，割り込み発生の原因によってベクトルを変化させて送り出しますので，処理ルーチンの組み立てが簡単にできます．

このモード2の割り込み機能はZ-80の大きな特徴です．ファミリと合わせてシステムを構成すれば，外部回路なしで優先順位の決定を含めた，割り込みシステムが完成してしまいます．







このような表を割り込み処理ルーチン・エントリー・アドレス・テーブルと呼ぶ

39 デージーチェーン

割り込みの優先順位の決定にはデージーチェーン（ひな菊の輪）という手法がとられます．CPU に対して割り込み信号を出すペリフェラルを直列に継いでおき，割り込みをかけて処理ルーチン実行中のペリフェラルが，自分より下位のペリフェラルに対して割り込み禁止を順次伝達していきます．自分より上位の割り込みがかかると，自分の処理が保留されますので，これもデージーチェーンを通じて知ることができます．処理が終ってリターン フロム インタラプト〔RETI〕命令を CPU がフェッチすると，データバスを横から監視している処理中のペリフェラルは自分の処理が終ったことを知り，下位のペリフェラルに対して割り込みの禁止を解除します．

各ペリフェラルはインタラプト イネーブル イン（IEI）とインタラプト イネーブル アウト（IEO）信号の端子を持っており，IEOを自分より下位のIEIに継ぎます．最も優先順位の高いペリフェラルのIEIは V_{CC}（＋5 V）へ継ぎ，最も低いペリフェラルの IEO はオープンにしておきます．

多数のペリフェラルを継ぐときは，伝達に時間がかかりますので，工夫がいります．そのままで継げるのは 4 個までです．

この方式は，割り込みだけでなく，マルチCPUとしたときや，DMAを複数使用するときのバスリクエストでも使われます．

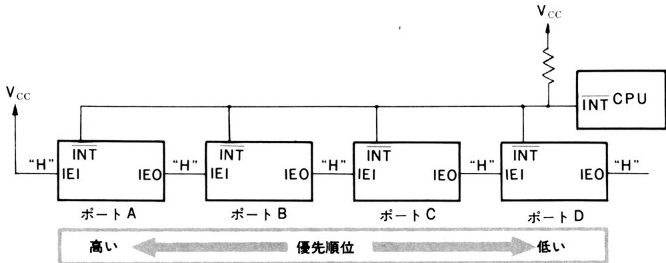
デージーチェーン端子（入力＝IEI　出力＝IEO）の機能

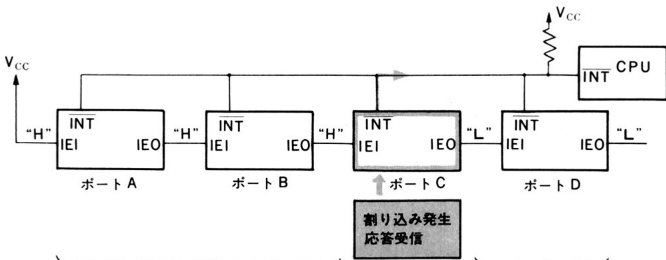
① 通常 IEI が“H”になっていれば IEO も“H”が出ている．

② 割り込みを発生し（$\overline{INT}$）応答サイクルに入ると，割り込みを発生した周辺は IEO を“L”にする．

③ IEI に“L”が入ると IEO も“L”にする．

④ 割り込みは，IEI が“H”になっているときだけ発生し，“L”のときは発生しない．またすでに発生した後に IEI が“L”になったときは保留されたものとする．





このときも割り込み要因があれば CPU に対して割り込みを発生できる．
このときは優先順位の低いポートは割り込み中であっても保留させられ，高いほうを先に処理する

割り込み処理終了の命令〔RETI〕をCPUがフェッチしたことをデータバスを見ていて知ると，IEO を“H”にもどす

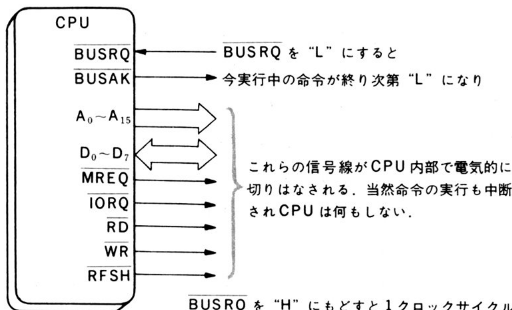
これより優先順位の高いポートが割り込み中なのでおわるまで待たされる

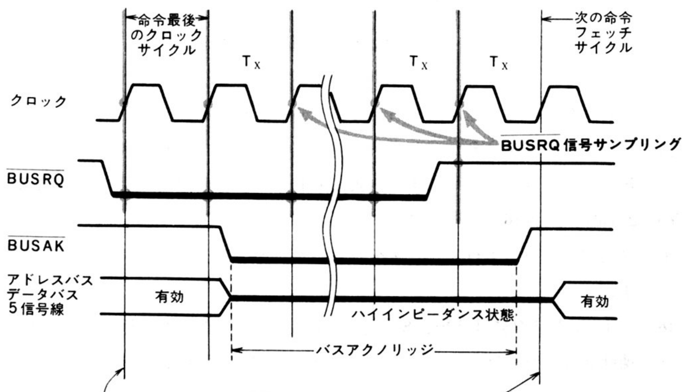
40 バス要求と応答

通常CPUはバスを占有して動作していますが，マルチCPUシステムで他のCPUからの要求や，その他いろいろな理由で一時動作を中止してバスを開けたいことがあります．このようなとき**バスリクエスト**（$\overline{\text{BUSRQ}}$）**信号**を与えると，トライステートの端子をすべて高インピーダンスにし，**バスアクノリッジ**（$\overline{\text{BUSAK}}$）**信号**をアクティブにして外部に対して，バスを開け渡したことを知らせてきます．この間メモリリフレッシュは行なわれなくなりますので，長時間この状態を続けるときは注意が必要です．

バスリクエスト信号は，割り込み信号と同様に，命令の最後のクロックサイクルの立ち上りの時点で検知されます．ここでバスリクエストが"L"であると，次の命令のフェッチサイクルはなくなり，バスアクノリッジがアクティブになります．これ以後各クロックの立ち上りでもバスリクエストが監視され"H"に戻っていると，次のクロックサイクルから元の動作に戻ります．バスアクノリッジがアクティブな期間は，ノンマスカブルインタラプトとインタラプトは受け付けられなくなります．しかしバスリクエストが最初に検知される時点での両割り込みは受け付けられ，バスアクノリッジが解除後に，それぞれの動作に入ります．

バスリクエスト端子を使用しないときは，V_{CC}へ継いで"H"レベルにしておいて下さい。


CPU
BUSRQ
BUSAK
$A_0 \sim A_{15}$
$D_0 \sim D_7$
MREQ
IORQ
RD
WR
RFSH
BUSRQ を "L" にすると
今実行中の命令が終り次第 "L" になり
これらの信号線がCPU内部で電気的に切りはなされる．当然命令の実行も中断されCPUは何もしない．
BUSRQ を "H" にもどすと1クロックサイクルの後次の命令から続行する．


命令最後のクロックサイクル
次の命令フェッチサイクル
T_X
T_X
T_X
クロック
BUSRQ 信号サンプリング
BUSRQ
BUSAK
アドレスバス
データバス
5信号線
有効
ハイインピーダンス状態
有効
バスアクノリッジ
ここまで入っていたNMIはバスリクエスト解除後に実行する．
バスアクノリッジ中はNMI，INTは無視される．
リフレッシュもされない．

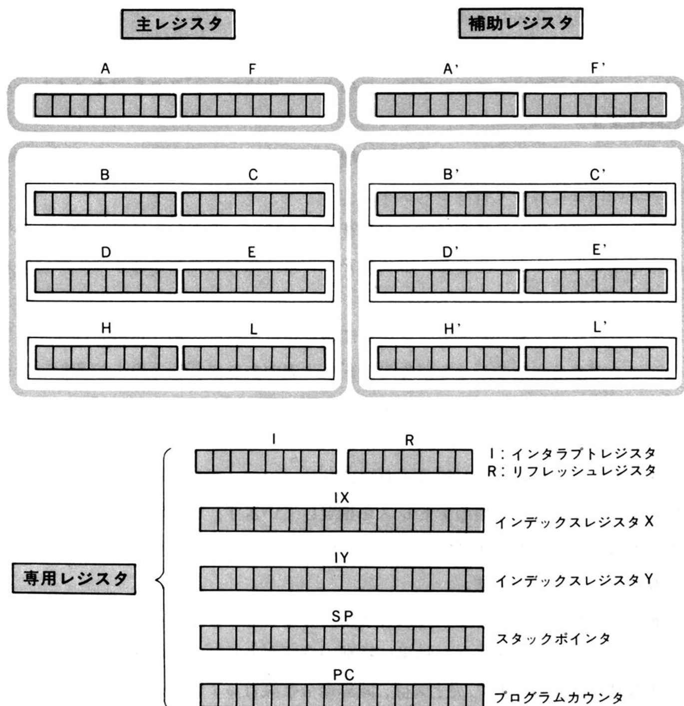
41 内部レジスタの構成

CPU内部には，種々の作業に使うための一時的なメモリがたくさんあります．1ビット単独のものは，フリップフロップ（F/F）と呼ばれ，複数ビットが一連にして使われるものは，レジスタと呼ばれます．このうちでユーザに公開されるレジスタが22本あります．それぞれに個性を持っていて，うまく使いわけることが，よいプログラムを作る上で重要です．どのレジスタをどう使うかは一概にはいえず，ケースバイケースで考えなければなりません．プログラムをいくつも作るうちに，だんだんとうまくなっていくでしょう．

8ビット単独で働くレジスタは，AレジスタとFレジスタです．Aレジスタはアキュムレータとも呼ばれ，最も重要な働きをします．Fレジスタはフラグレジスタとして各ビットに意味を持つレジスタです．8ビット単独でも，二つを継いで16ビットとしても働くレジスタは，B，C，D，E，H，L，です．ここまでの8本のレジスタは，主と補助の2組あり，内容を交換することができます．

専用レジスタとして用途が決められているものは6本あります．割り込み処理で使われるIレジスタは8ビット，メモリ リフレッシュ アドレスをカウントしているRレジスタは7ビットです．作表に便利なインデックス レジスタはIX，IYの2本あり，各16ビットです．スタックのアドレスを入れておく スタック ポインタとプログラム実行アドレスをカウントするプログラム カウンタも16ビットです．

16ビット レジスタには2桁，8ビット レジスタには1桁の名称が付けられており，補助レジスタには’（ダッシュ）を付けて，たとえばA’と呼ぶようになっています．



主レジスタ
補助レジスタ
A
F
A'
F'
B
C
B'
C'
D
E
D'
E'
H
L
H'
L'
専用レジスタ
I
R
I：インタラプトレジスタ
R：リフレッシュレジスタ
IX
インデックスレジスタX
IY
インデックスレジスタY
SP
スタックポインタ
PC
プログラムカウンタ

42 A, I, R, Fレジスタ

レジスタ群の中で最も多用されるAレジスタは，人が手作業で計算するときのソロバンにたとえられます．加算を行なう場合を例にとると，まず足される数をAレジスタへ入れます．次にAレジスタに対して，別の足す数を加えると，Aレジスタの内容が答になるわけです．このような用途のレジスタを**アキュムレータ**と呼びます．

8ビットの加算，減算，論理和，論理積，排他的論理和，比較は，すべてAレジスタの内容に対して行なわれます．

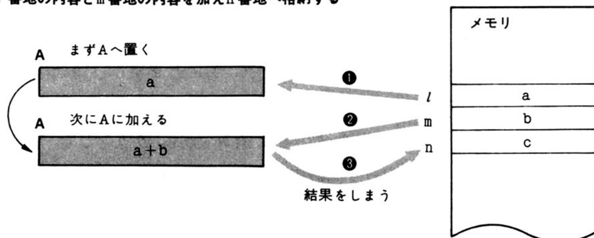
IレジスタとRレジスタに読み書きするときは，一度Aレジスタを経由しないとできません．入力，出力も，ポートアドレスに絶対番地を指定するときは，Aレジスタを経由します．

Iレジスタは，割り込みモードを2に指定したときに，割り込み処理ルーチンへジャンプするためのジャンプ先のアドレス表のあるアドレスの上位8ビットを表わします．初期値設定のために1回書き込むだけで，ほとんどいじる必要はないレジスタです．

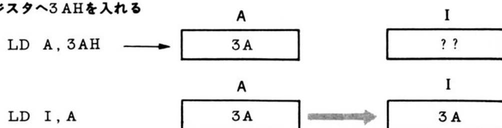
Rレジスタは，リセット信号でゼロになり，フェッチサイクルごとに1ずつ増加して，リフレッシュ用のアドレスを指定しています．プログラムで内容を変えることはほとんどありません．むしろ正確にリフレッシュさせるためには，さわらないほうがよいのです．

Fレジスタは，プログラムで内容を変えることはできません．1命令実行ごとに，その命令により出てきた結果に特別の意味があれば，それ以後の命令実行に必要な情報を記憶します．たとえば，減算命令を実行して結果がゼロであれば，Fレジスタ中の7番目のビット（これをゼロフラグといいます）を"1"にしてこのことを憶えておきます．次の命令が，「結果がゼロならばジャンプせよ」といった命令である場合，これをチェックして判断するようになっています．このように種々のフラグ（旗）が並んでいるレジスタがFレジスタです．

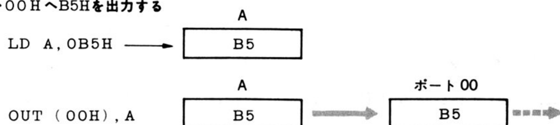
●*l* 番地の内容とm番地の内容を加えn番地へ格納する



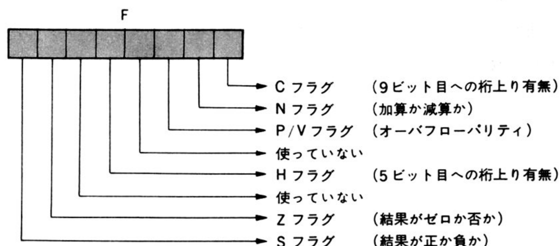
●Iレジスタへ3AHを入れる



●ポート00HへB5Hを出力する



●Fレジスタ



43 汎用レジスタ

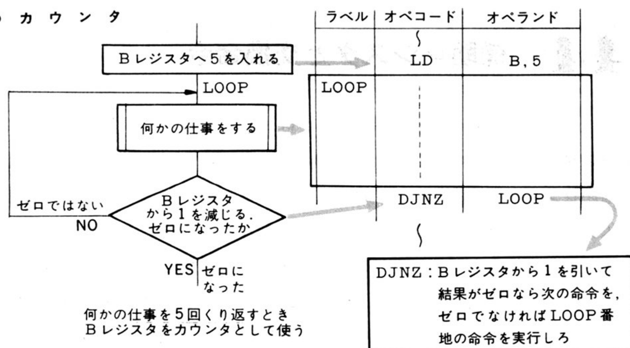
汎用レジスタには，H，L，B，C，D，Eの6本と，同数の補助レジスタがあります．2本のレジスタをペアにして，HL，BC，DEと表現し，16ビット レジスタとして使うこともできます．

8ビット レジスタとしての働きは，主に演算途中の数値などを一時記憶することですが，Bレジスタはループ回数を入れておき，1回に1ずつ減算し，ゼロになったら，ループから出るというカウンタの役目をさせることがあります．このためにデクリメント ジャンプ ノンゼロ〔DJNZ〕命令がとても便利です．Cレジスタは入力〔IN〕，出力〔OUT〕命令で，ポート アドレスを指定するポインタとして使えます．

16ビット レジスタとしてペアにして使うと，HLは加算減算でアキュムレータとして働きます．また，アドレスを入れ，間接的にアドレス指定をすることができます．

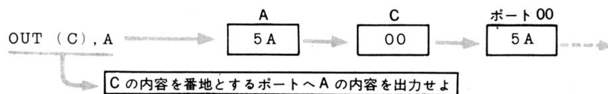
繰返し処理でアドレスを入れておくレジスタを**ポインタ**，回数を入れておくレジスタを**カウンタ**と呼びます．ブロック転送〔LDIR〕命令は，元のアドレスをHL，先のアドレスをDE，ブロックの長さ（バイト数）をBCに入れてから実行すると，HLの内容番地のメモリからDEの内容番地のメモリへ1バイト転送し，HLとDEに1を加え，BCから1を引きます．これを繰り返しBCがゼロになると実行を終る，という命令です．HLとDEをポインタとして，BCをカウンタとして使用しています．

● カウンタ

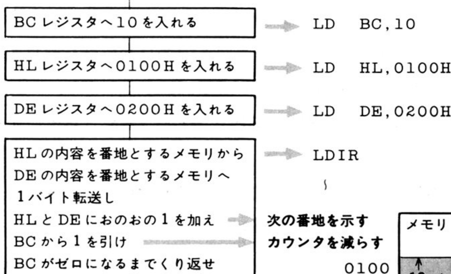


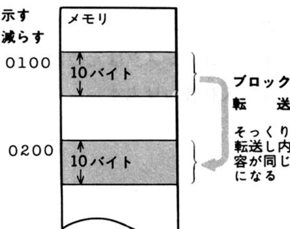
● ポインタ

Cレジスタの内容が　00H
Aレジスタの内容が　5AH　とすると



● ポインタとカウンタ





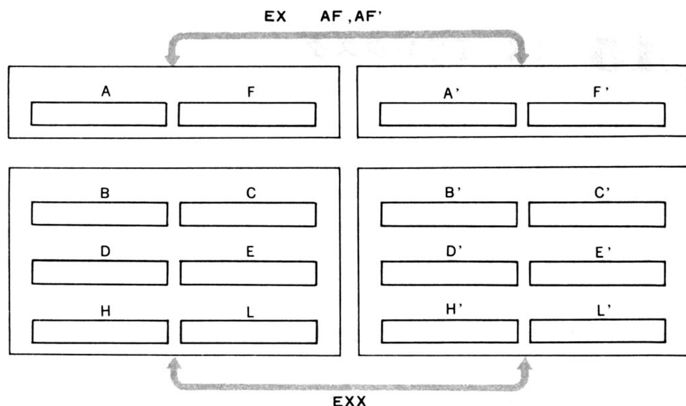
44 補助レジスタと交換命令

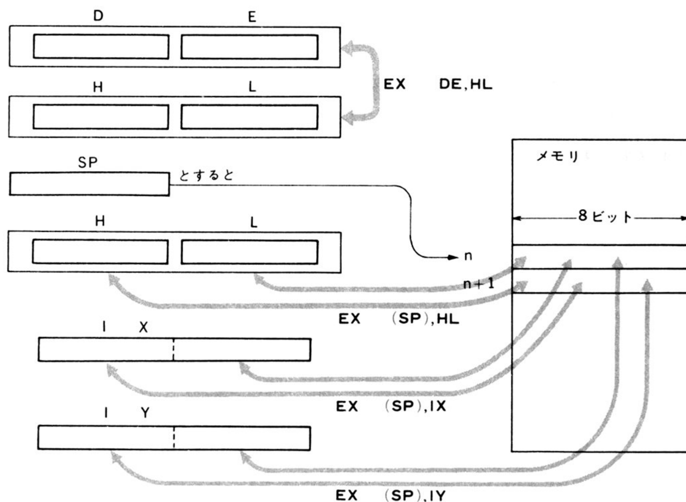
A, F, H, L, B, C, D, Eの各レジスタは，同じものがもう1組あります．**補助レジスタ**と呼び，主レジスタとは区別して，“ ’ ”ダッシュをつけて表現します．補助レジスタの内容は，交換命令と呼ばれる一連の命令で，主レジスタの内容と入れ替える以外に操作することはできません．したがって，主レジスタの裏側にあって，交換命令でひっくり返すと，表（主）へ出てくると考えられます．補助レジスタとの交換命令には，AFとAF’のペアを交換する〔EX AF, AF’〕と，上記以外の汎用レジスタを交換する〔EXX〕の二つがあります．

サブ ルーチンや割り込み処理ルーチンの中でメイン ルーチンで使用中のレジスタの内容を変えたくないことがあります．このようなときは，サブ ルーチンに入ったところで，レジスタの内容をメモリへ待避させてから，必要なレジスタを使い，サブ ルーチンから出る前に待避した内容をレジスタへ戻せばよいのですが，交換命令を使えば，1命令で全レジスタを待避させることができます．特に，スピードが問題になる割り込み処理の場合は都合のよい命令です．ただし，メイン ルーチンで主レジスタ，サブ ルーチンで補助レジスタを使うことをきちんと取り決めておかないと，どちらに何が入っているのかわからなくなります．

交換命令には他にDEとHLを交換する〔EX DE, HL〕と，HL, IX, IYの各内容とスタッカの内容を交換する〔EX (SP), HL〕,〔EX (SP), IX〕,〔EX (SP), IY〕があります．

なお，通常の転送命令の場合は，たとえば〔LD A, B〕であれば，Bレジスタの内容がAレジスタへ転送されるだけで，元のAレジスタの内容はなくなってしまいます．Bレジスタは元のまま変わりません．交換命令では双方ともこわれることなく入れ替わります．


EX AF,AF'
A
F
A'
F'
B
C
D
E
H
L
B'
C'
D'
E'
H'
L'
EXX
EX, EXX 以外の命令では
主レジスタしか扱えない


D
E
H
L
EX DE,HL
SP
とすると
メモリ
8ビット
H
L
n
n+1
EX (SP),HL
I X
EX (SP),IX
I Y
EX (SP),IY

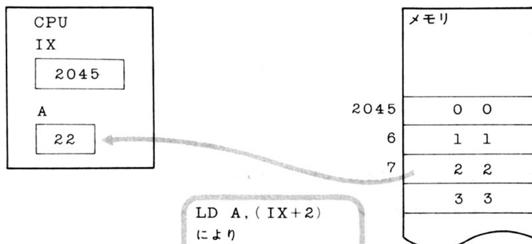
45 IX, IYレジスタ

IX，IYレジスタは，**インデックス レジスタ**と呼ばれ，同一の働きをする16ビット レジスタが2本あります．通常はここへメモリのアドレスを入れて，間接的にアドレス指定をするときに使います．表を操作するときなどに表の先頭アドレスを入れておき，1ずつ増加させながら次々に読み出すことができます．また，先頭から何番目のデータが欲しいというようなときは，先頭からの距離をプログラムで指定することもできるレジスタです．たとえば，〔LD A,（IX+5）〕と書けば，IXに入っている数プラス5番地のメモリ内容をAレジスタへ入れることができます．距離は，－128～＋127バイトまでの範囲に限られます．距離のことを**ディスプレイスメント**と呼びます．

このレジスタの内容を間接アドレスとしてソースまたはディスティネーションに指定した命令のマシン語は，オペコードが2バイト構成になります．3バイト目には，ディスプレイスメントの値が入ります．オペランドがある場合は4バイト目に続きます．

実行のサイクルは，ディスプレイスメントの演算のためのクロック・サイクルが入るため，例外的に長くなります．〔LD A,（IX+5）〕では，19クロックサイクルです．

たとえばIXレジスタに2045Hが入っていると



LD A,(IX+2)
により
AレジスタにはIX+2
すなわち2047H番地
の内容が転送される

LD A,(IX+2)のマシン語
DD オペコード
7E オペコード
02 ディスプレイスメント

	メモリ
2040	①
1	②
2	③
2043	①
4	②
5	③
2046	

メモリに3バイトを一組とするデータ
があるとする

```
        〜
        LD   IX,2040H      最初の組の先頭番地をIXへ
        LD   DE,3          DEへ3を入れておく
LOOP    LD   A,(IX)        1バイト目をAレジスタへ
        LD   B,(IX+1)      2バイト目をBレジスタへ
        LD   C,(IX+2)      3バイト目をCレジスタへ
        〜
             A,B,Cレジスタにある
             1組を処理する
        ADD  IX,DE         次の組の先頭番地をIXへ
        JP   LOOP
```

46 スタッカとスタックポインタ（SPレジスタ）

サブルーチンコール〔CALL〕命令を実行したり，割り込みルーチンへジャンプしたりするときに，戻り番地を自動的に記憶する機能があることは前述しました．これ以外にもプログラム中で一時的にレジスタの内容をメモリに退避したいことがよくあります．このようなときに大変便利なのがスタッカです．

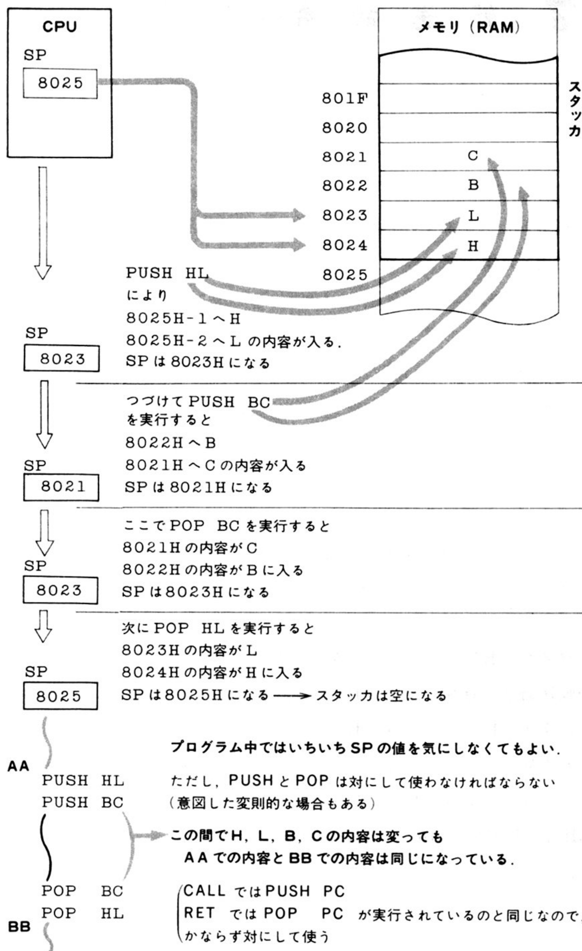
スタッカはメモリ（RAM）の一部を割りあてた記憶領域のことですが，このアドレスは，**スタックポインタ**（SP）と呼ばれる16ビットレジスタに入れておきます．

スタックポインタに8025Hという値がプログラムによって入れられているとしましょう．このとき〔PUSH HL〕を実行すると，8025H-1の8024HへHの内容が，8025H-2の8023HへLの内容が書き込まれスタックポインタは8023Hになっています． 次に読み出すときは〔POP HL〕を実行すると，8023Hの内容をLに，8023H+1の内容をHに転送し，スタックポインタは8025Hになります．

スタッカに次々にしまい込まれた値は後から入れたほうが先に読み出されますので，First In Last Out（FILO）タイプのスタッカと呼ばれます．サブルーチンや割り込みルーチンに入るときは自動的に〔PUSH PC〕* に相当する命令が実行され，戻るときは〔POP PC〕* に相当する命令が実行されていることになります．したがってサブルーチンの中から次のサブルーチンをコールするときも，（これをサブルーチンの**ネスティング**といいますが）うまく働いてくれます．ただし，スタッカとして割りあてた領域を超えるネスティングはできません．未完成のプログラムであやまって次々にサブルーチンコールをすると，このようなことが起こることがあります．プログラム設計の際に，スタッカの大きさを決定するときは，慎重な判断を要します．またプログラムの始まりでは必ずスタックポインタのイニシャライズ（初期値設定）を行なって下さい．〔PUSH〕，〔POP〕できるペアレジスタはAF，BC，DE，HL，IX，IYです．

---

* この命令は，プログラム上では使えない．


CPU
SP
8025
メモリ (RAM)
スタッカ
801F
8020
8021
8022
8023
8024
8025
C
B
L
H
PUSH HL
により
8025H-1へH
8025H-2へLの内容が入る.
SPは8023Hになる
SP
8023
つづけてPUSH BC
を実行すると
8022HへB
8021HへCの内容が入る
SPは8021Hになる
SP
8021
ここでPOP BCを実行すると
8021Hの内容がC
8022Hの内容がBに入る
SPは8023Hになる
SP
8023
次にPOP HLを実行すると
8023Hの内容がL
8024Hの内容がHに入る
SPは8025Hになる ──→ スタッカは空になる
SP
8025
プログラム中ではいちいちSPの値を気にしなくてもよい.
AA
PUSH HL
PUSH BC
ただし, PUSHとPOPは対にして使わなければならない
(意図した変則的な場合もある)
この間でH, L, B, Cの内容は変っても
AAでの内容とBBでの内容は同じになっている.
POP BC
POP HL
BB
CALLではPUSH PC
RET ではPOP PC が実行されているのと同じなので,
かならず対にして使う

47 転送命令

最も基本的な転送命令は，ロード命令〔LD a,b〕です．オペランドにはソースとディスティネーションの指定をします．ニモニックはZ-80ではすべて〔LD〕で統一されています．8ビットの転送では，ソースにはレジスタ名，ペアレジスタによる間接アドレス，インデックス レジスタによるディスプレイスメント，直接アドレス値および定数が指定できます．ディスティネーションには，ソースで使用できるうちの定数以外を指定できます．ただし，I，Rレジスタは Aレジスタとのやりとりしかできないなど，ソースとディスティネーションの組み合わせには限定があります．

16ビット転送では，ペア レジスタとやりとりのできる組み合わせは，16ビット定数，直接アドレス値だけで，ペア レジスタ間のやりとりは，HL，IX，IYからスタック ポインタ（SP）への転送だけになります．

アセンブリ語で記述する場合，アドレス値や定数は，10進数，16進数で書いてもよいのですが，ラベル名で記述することもできます．マシン語になおしたときは，アセンブルリスト上は，16進数に統一されます．16ビット定数やアドレス値のときは，オペコードの次に下位8ビット，その次に上位8ビットがくるように並びます．

レジスタ名や定数に（　）カッコを付けたオペランドは，メモリへまたはメモリからの転送命令を意味します．カッコ内の数値やラベルは，メモリのアドレスを指しています．16ビット転送の場合は，このアドレスが下位8ビット，次が上位8ビットの転送アドレスになり，ペア レジスタとの転送では，たとえばHLならば，Hレジスタが上位，Lレジスタが下位に対応します．

〔PUSH〕，〔POP〕も16ビット転送命令ですが，スタック ポインタの項で述べました．

8ビットロード

ディスティネーション（受け側）		ソース（送り側）		
		レジスタ	メモリ	定数
レジスタ	A	A, B, C, D, E, H, L, I, R	(HL), (BC), (DE), (IX+d), (IY+d), (定数)	定数
	B C D E H L	A, B, C, D, E, H, L	(HL), (IX+d), (IY+d)	定数
	I R	A	—	—
メモリ	(HL)	A, B, C, D, E, H, L	—	定数
	(BC) (DE)	A	—	—
	(IX+d) (IY+d)	A, B, C, D, E, H, L	—	定数
	(定数)	A	—	—

アドレス　　Aレジスタはすべての転送ができる　　メモリ→メモリの転送はできない　　8ビット定数

16ビットロード

ディスティネーション（受け側）		ソース（送り側）		
		レジスタ	メモリ	定数
レジスタ	BC DE HL	—	(定数)	定数
	SP	HL, IX, IY	(定数)	定数
	IX IY	—	(定数)	定数
メモリ	(定数)	BC, DE, HL, SP, IX, IY	—	—

PUSH, POPのできるレジスタ
AF, BC, DE, HL, IX, IY

LD (nn), HLや
LD HL, (nn)
の場合
nn　　へL（下位）
nn+1へH（上位）
が対応する

メモリ	
nn	L
nn+1	H

48 算術演算命令

レジスタやメモリ内の8ビットないし16ビットのビットパターンを2進数と見なして，数学的な演算をする命令があります．8ビットの演算は，アド〔ADD〕，サブトラクト〔SUB〕，アド ウィズキャリ〔ADC〕，サブトラクト ウィズ キャリ〔SBC〕の4命令があります．Aレジスタに対して，他のレジスタかメモリの内容を加減算します．この演算のとき桁あふれまたは桁がりをする命令と無視する命令に別れます．あふれた桁はFレジスタの1ビット目のキャリフラグ（Cフラグ）があてられます．

コンペア〔CP〕命令は，Aレジスタからオペランドの内容を減算しますが，Aレジスタの内容は変わらず，Fレジスタが減算のときと同じに変化し，あとの命令で，結果がゼロであったかチェックできます．ゼロであったときは，Aレジスタの内容とオペランドの内容が同じであると判断できます．

インクリメント〔INC〕命令は，オペランドに1を加える命令です．デクリメント〔DEC〕は，オペランドから1を引く命令です．メモリ内のデータ列を1バイトずつ順次操作する場合や繰り返し行なう仕事の回数をカウントするときに使います．

ニゲイト〔NEG〕は，Aレジスタの内容をゼロから引いてAレジスタに入れる命令です．すなわち，Aレジスタの内容の符号（プラス，マイナス）を反転させるとき主に使います．

16ビットの演算は，HL，IX，IYレジスタの内容に対して行なう加減算で，桁あふれを考えに入れない演算は加算だけです．インクリメント〔INC〕とデクリメント〔DEC〕は，8ビットと同様です．メモリアドレスの操作に使います．

算術演算命令でのFレジスタの働きは重要で，多桁の演算では，キャリフラグの働きに注意する必要があります．また10進補正〔DAA〕命令は，Fレジスタの結果を参照して補正のし方を決定します．

	命令	ディスティネーション 被演算数→答	ソース			操作 s：ソース Cy：キャリ
			レジスタ	メモリ	定数	
8ビット	ADD	A	A, B, C, D E, H, L	(HL) (IX+d) (IY+d)	定数	A+s→A
	ADC	A				A+s+Cy→A
	SUB	A*				A−s→A
	SBC	A				A−s−Cy→A
	CP	A*				A−s　A は不変
	INC	ソースと同じ*	A, B, C, D E, H, L	(HL) (IX+d) (IY+d)	—	s+1→s
	DEC	ソースと同じ*				s−1→s
	DAA	A*	A*	—	—	A を補正
	NEG	A*				0−A→A
16ビット	ADD	HL	BC, DE HL, SP	—	—	HL+s→HL
	ADD	IX	BC, DE SP, IX			IX+s→IX
	ADD	IY	BC, DE SP, IY			IY+s→IY
	ADC	HL	BC, DE HL, SP			HL+s+Cy→HL
	SBC	HL				HL−s−Cy→HL
	INC	ソースと同じ*	BC, DE, HL SP, IX, IY			s+1→s
	DEC	ソースと同じ*				s−1→s

*印はアセンブリ語ではディスティネーションはオペランドに書かない．(例) SUB C　　INC A
DAA NEG ではソースも書かない．

ADD　ADC

ADC では C フラグ(キャリ)もここへ加える．すなわち，さらに下の桁の演算がこの前に行なわれているとすると，そのときの桁上りが含まれる

```
   10111001
+)01010101
 100001110
```

桁上り　C フラグ（キャリ）へ入る

SBCは，けたがりがCフラグに入っているのでこれを含めての演算をする
SUBは，前の演算でけたがりがなかったものとしてCフラグは無視してしまう

49 論理演算命令

論理演算命令は，すべて8ビット単位に，Aレジスタに対して行なわれます．各ビットごとの論理積，論理和，排他的論理和，否定をとります．

アンド〔AND〕命令は論理積です．Aレジスタとオペランドの内容の双方共に"1"のビットだけ"1"が残り，他は"0"になります．Aレジスタのある特定のビットだけ残したいときは，ここを"1"，他を"0"としたビットパターンをオペランドで指定して，アンドをとれば，不要なビットは必ず"0"になり，必要なビットだけが"1"であれば"1"，"0"であれば"0"として残ります．この方法を**マスク**といいますが，Z-80ではビット操作命令があるので，あまり使いません．

オア〔OR〕命令は論理和です．Aレジスタとオペランドの内容のいずれかが"1"のビットを"1"として，双方共に"0"のビットを"0"にします．16ビット演算のデクリメント〔DEC〕命令でHLレジスタの内容を1ずつ減らしてゼロになってもフラグは変化しません．そこで，このチェックを行なうときは，HレジスタをAレジスタに移して，Lレジスタとのオア〔OR〕をとりますと，H，L，共に全ビットが"0"すなわちHLレジスタとしてゼロになっているときだけ結果がゼロになり，フラグレジスタのゼロフラグがこのことを示します．また，特定のビットを前の内容にかかわらず"1"にしたいときには，このビットが"1"，他が"0"のビットパターンとオア〔OR〕をとれば，"1"のところは"1"になり，"0"のところは前のまま残ります．

エクスクルーシブオア〔XOR〕は排他的論理和です．Aレジスタとオペランドの内容が，ビット単位に同じところを"0"，異なるところを"1"にします．AレジスタとAレジスタのエクスクルーシブオアをとるとAレジスタは必ずゼロになります．なおキャリフラグだけを"0"にするときは，〔AND A〕か〔OR A〕を使います．

コンプリメント〔CPL〕命令はビットパターンをすべて反転させます．すなわち"1"のビットは"0"に，"0"のビットは"1"にする論理否定です．

命令	ディスティネーション	ソース		
		レジスタ	メモリ	定数
AND OR XOR	A	A, B, C, D, E, H, L	(HL), (IX+d), (IY+d)	定数
CPL	A	A	—	—

アセンブリ言語ではディスティネーションはオペランドに書かない.
CPL ではソースも書かない.

AND

a	b	結果
0	0	0
0	1	0
1	0	0
1	1	1

(例)								
Aレジスタ	0	1	0	0	1	0	1	1
ソース	1	1	0	1	0	0	0	1
Aレジスタ(結果)	0	1	0	0	0	0	0	1

OR

a	b	結果
0	0	0
0	1	1
1	0	1
1	1	1

(例)								
Aレジスタ	0	1	0	0	1	0	1	1
ソース	1	1	0	1	0	0	0	1
Aレジスタ(結果)	1	1	0	1	1	0	1	1

XOR

a	b	結果
0	0	0
0	1	1
1	0	1
1	1	0

(例)								
Aレジスタ	0	1	0	0	1	0	1	1
ソース	1	1	0	1	0	0	0	1
Aレジスタ(結果)	1	0	0	1	1	0	1	0

CPL

前	結果
0	1
1	0

(例)								
Aレジスタ	0	1	0	0	1	0	1	1
Aレジスタ(結果)	1	0	1	1	0	1	0	0

50 ビット操作命令

レジスタ，メモリ内のどこの1ビットでも，ビット操作命令で"0"か"1"にすることができ，また"0"か"1"かの判定をすることができます．コントローラ的な応用にはよく使う命令で，Z-80の特徴の一つといえます．

セット〔SET〕命令は，ビットを"1"にする命令で，オペランドにはビットの位置とレジスタ名か，メモリのアドレスを示すレジスタHL，IX，IYのいずれかを指定します．ビットの位置は，2のn乗のnで表わしますので，右端が0，左端が7となります．

リセット〔RES〕命令は，ビットを"0"にする命令です．オペランドはセット命令と同じです．

ビット〔BIT〕命令は，指定のビットを調べ，"0"であればFレジスタのゼロフラグ（Zフラグ）を"1"にして，"0"であったことを記憶しておきます．後に書かれた条件付ジャンプ命令で，"0"のときと"1"のときの飛び先番地を別々に指定してあれば，条件に応じた手順で仕事を進めます．オペランドはセット命令と同じです．

23A6H番地のメモリの右から3番目のビットを"1"にするときは，HLレジスタに23A6Hをロード命令で書き込んでから〔SET 2,（HL）〕を実行させ，"0"にするときは〔RES 2,（HL）〕，内容を判定するには〔BIT 2,（HL）〕となります．

ヨシミ

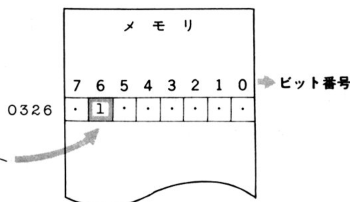
命令	ディスティネーション	ソース			
		レジスタ	メモリ		ビット
SET RES BIT	ソースと同じ	A, B, C, D, E, H, L	(HL), (IX+d), (IY+d)		0〜7

アセンブリ言語ではディスティネーションはオペランドに書かない.

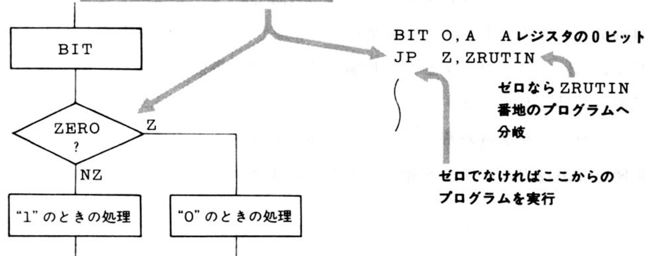
HL

0326

のとき
SET 6,(HL)
を実行すると
メモリの
0326H番地の第6ビット
が"1"になる



SET　"1"にする
RES　"0"にする
BIT　指定のビットが"0"か"1"かによって
Fレジスタの Z フラグを "0"ならZ(ゼロ)
"1"ならNZ(ノンゼロ)を示すように設定する
この場合, 後にある判定命令(ジャンプ命令)でZフラグの状態を判定し以後の処理を分岐させる.



51 ローテート，シフト命令

ローテート命令は，ビットパターンを右か左に一つだけずらす命令です．はみ出したビットは最後に回り込んでつながります．このときキャリフラグを含めて回転するか，含めずに回転し，回り込んだビットと同じものをキャリフラグへ入れるかによって大別されます．ライト ローテート〔RR〕，レフト ローテート〔RL〕，ライト ローテート サーキュラ〔RRC〕，レフト ローテート サーキュラ〔RLC〕があり，特にAレジスタに対する命令は，8080系のCPUと互換性を持たせるために，1バイト命令が別に用意されています．これ以外のローテートとシフトはZ-80特有の機能です．

ライト（レフト）ローテートデジット〔RRD〕，〔RLD〕命令は，HLレジスタの内容で示されるメモリとAレジスタの下4ビットとの間で4ビット（1デジット）を一まとめにして回転させる命令で，2進化10進数の多桁演算によく使われるものです．

シフト命令は，はみ出したビットを切りすてて，最後には"0"または左端ビットの値（符号ビット）が入ります．シフト ライト アリスメチック（算術的右シフト）〔SRA〕，シフト レフト アリスメチック〔SLA〕，シフト ライト ロジカル〔SRL〕の三つです．算術的シフトは，ビットパターンを2進数とみなしたときシフトしても符号が変わらないよう配慮されているシフトです．左シフトは算術的（アリスメチック）も論理的（ロジカル）も同じですから一つしかありません．左に一つシフトすると，数値は2をかけた値になり，右にシフトすると2で割った値になります．

ビットパターンを2進数扱いでかけ算，割り算をするときは，シフトとローテートを使って高速のルーチンを組むことができます．

	命令	ソース（ディスティネーション）		動作
		レジスタ	メモリ	
ローテート	RLCA	A*	—	
	RLC	A,B,C,D E,H,L	(HL) (IX+d) (IY+d)	Cy ← 7 ← 0
	RLA	A*	—	
	RL	A,B,C,D E,H,L	(HL) (IX+d) (IY+d)	Cy ← 7 ← 0
	RRCA	A*	—	
	RRC	A,B,C,D E,H,L	(HL) (IX+d) (IY+d)	Cy ← 7 → 0
	RRA	A*	—	
	RR	A,B,C,D E,H,L	(HL) (IX+d) (IY+d)	Cy → 7 → 0
	RLD	A+(HL)*		A 7 4 3 0　(HL) 7 4 3 0
	RRD	A+(HL)*		A 7 4 3 0　(HL) 7 4 3 0
シフト	SLA	A,B,C,D E,H,L	(HL) (IX+d) (IY+d)	Cy ← 7 ← 0 ← "0"
	SRA	A,B,C,D E,H,L	(HL) (IX+d) (IY+d)	Cy 7 → 0
	SRL	A,B,C,D E,H,L	(HL) (IX+d) (IY+d)	Cy "0" → 7 → 0

＊印はアセンブリ言語ではオペランドに記述しない

Cy はＦレジスタのＣフラグ（キャリ）

SRAの7番ビットは変わらない

シフトではCyの元の値は失われる

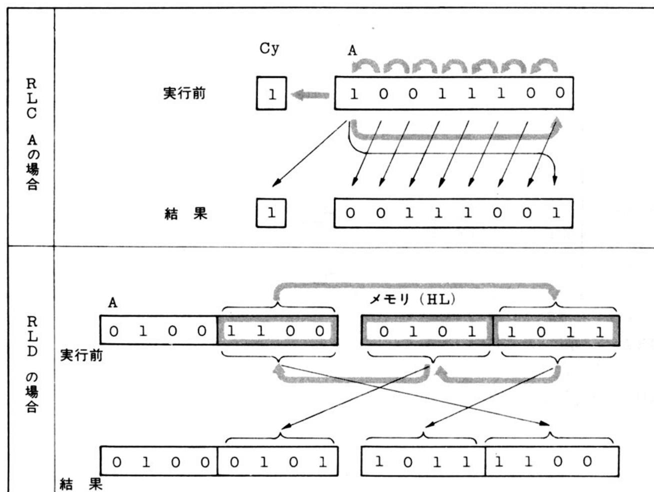
RRD，RLDでは移動する4ビット単位の内容は変わらない

（例）

Cy　A

1　10011100　のときの結果は

命　令	Cy	A
RLC　A	1	00111001
RL　A	1	00111001
RRC　A	0	01001110
RR　A	0	11001110
SLA　A	1	00111000
SRA　A	0	11001110
SRL　A	0	01001110



シフト，ローテート を使ったプログラム例

（16ビットのかけ算サブルーチン）

メモリに図のようにaとbの値が2進数で入っている．

a×b＝c

の演算をし，答cを32ビットとして，メモリに格納する．

メモリのアドレスnはIXレジスタに入れられているとする．

```
MPX   LD    C,(IX+0)
      LD    A,(IX+1)
      LD    E,(IX+2)
      LD    D,(IX+3)
      LD    B,16
      LD    HL,0
      EXX
      LD    HL,0
      LD    DE,0
      EXX
LOOP  SRL   A
      RR    C
      JR    NC,JNDT
      ADD   HL,DE
      EXX
      ADC   HL,DE
      EXX
JNDT  SLA   E
      RL    D
      EXX
      RL    E
      RL    D
      EXX
      DJNZ  LOOP
      LD    (IX+4),L
      LD    (IX+5),H
      EXX
      LD    (IX+6),L
      LD    (IX+7),H
      RET
```

	メモリ
n	a（下　位）
	a（上　位）
	b（下　位）
	b（上　位）
	c（最下位）
	c（下　位）
	c（上　位）
	c（最上位）

52 ブロック転送，ブロックサーチ，ブロック入出力命令

メモリ内の複数バイトのブロックを別の番地へ移し変える場合，1バイトずつCPU内のレジスタへ読み込み，別の番地へ書き込む操作を必要バイト数になるまで繰り返します．また，メモリ内のブロックの中から指定のビットパターンと同じパターンを持つ1バイトを探し出すときも，1バイトずつAレジスタへ読み込んではコンペア〔CP〕命令を実行し，一致するまで繰り返します．入出力のときもメモリブロックの内容をポートへ次々に出力したり，次々にポートから読んだデータをメモリへ並べたりするときは繰り返しのプログラムを組まなければなりません．これを一つの命令で置き換えられるのが，この命令群です．

ブロック転送命令は，HLレジスタに元のアドレス，DEレジスタに転送先のアドレス，BCレジスタに転送バイト数をあらかじめ書いておき，次に転送命令を実行させますと，HLとDEを1ずつ進め，BCを1ずつ減らしていきます．

ブロックサーチ命令は，HLレジスタにアドレス，BCレジスタにブロックのバイト数を，比較すべきビットパターンをAレジスタに書いておきます．一致するとFレジスタのゼロフラグが，BCレジスタがゼロになる（一致するものがない）とオーバフローフラグ（Vフラグ）がリセットされますので，次の命令でフラグ判定をしなければなりません．

ブロック入出力命令は，HLにデータ格納のメモリアドレス，CレジスタにIOポートアドレス，Bレジスタにバイト数を書いてから実行させます．

これらの命令には，メモリブロックを先頭から扱うインクリメントグループと後から扱うディクリメントグループがあります．また，全データの操作が終る（サーチでは一致した場合も含む）までプログラムカウンタが変わらないで，自動的にループするリピートの機能を持つ命令もあります．転送，サーチ，入出力しながら，何かのデータ操作がない場合は，リピートは便利です．1バイト扱うごとにフェッチサイクルから始まります．

ブロック転送

（HL←HL+1の意味は，HLの内容に1を加えるということ）

命　令	動	作	
LDI	HLの内容を番地とするメモリから DEの内容を番地とするメモリへ転送する	HL←HL+1　BC←BC−1	1バイトだけ転送して終る
LDIR		DE←DE+1	BC=0までくり返す
LDD		HL←HL−1　BC←BC−1	1バイトだけ転送して終る
LDDR		DE←DE−1	BC=0までくり返す

BC≒0のときはVフラグがセットされる

ブロックサーチ

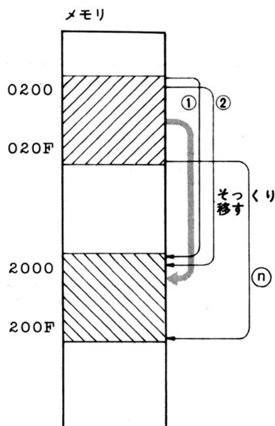
命　令	動		作	
CPI	HLの内容を番地とするメモリの内容とAレジスタの内容を比較する	A−(HL)=0…等しい A−(HL)≒0…等しくない 等しいときZフラグをセット（ゼロの状態）	HL←HL+1 BC←BC−1	1バイトだけ調べて終る
CPIR				BC=0か A−(HL)=0まで
CPD			HL←HL−1 BC←BC−1	1バイトだけ調べて終る
CPDR				BC=0か A−(HL)=0まで

BC≒0のときはVフラグ，A−(HL)=0のときはZフラグがセットされる

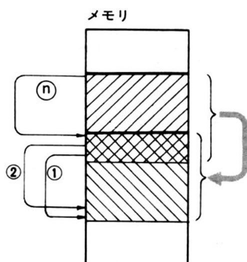
ブロック入出力

命　令	動		作	注
INI	Cの内容を番地とするポートからHLの内容を番地とするメモリへ読み込む	HL←HL+1	1バイト入力して終る	Cは変わらない カウンタはBだけ（256まで）
INIR		B←B−1	B=0までくり返す	
IND		HL←HL−1	1バイト入力して終る	
INDR		B←B−1	B=0までくり返す	
OUTI	HLの内容を番地とするメモリの内容をCの内容を番地とするポートへ出力する	HL←HL+1	1バイト出力して終る	
OTIR		B←B−1	B=0までくり返す	
OUTD		HL←HL−1	1バイト出力して終る	
OTDR		B←B−1	B=0までくり返す	

B=0のときはZフラグがセットされる




```
     LD   HL,0200H
     LD   DE,2000H
     LD   BC,0010H
LOOP LD   A,(HL)    ┐
     LD   (DE),A    │
     INC  HL        │=LDI   ┐
     INC  DE        │       │
     DEC  BC        ┘       │=LDIR
     LD   A,B    ┐ B,Cゼロである │
     OR   C      ┘ ことを調べる   ┘
     JP   NZ,LOOP ──▶ ノンゼロなら
                         LOOPへ
                  ◀── ゼロなら次へ
                      （終り）
```



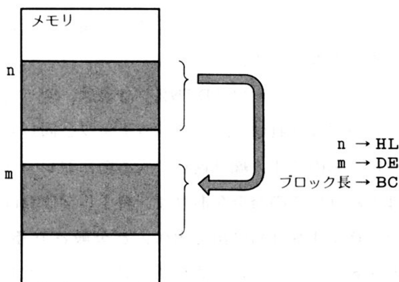
転送前と後のエリアが
重なっていると，最初のほうから転送した場合これから転送すべき内容の所へ書いてしまい，内容が正しく残らない．
このようなときは最後のほう（下）から転送する
＝LDDR命令を使う

ブロック転送命令を使ったプログラム例

（ブロック転送サブルーチン）

HLレジスタに転送前アドレス，DEレジスタに転送後アドレス，BCレジスタに転送するブロックのバイト数が入っているとする．転送前と転送後のエリアが重なっている場合を想定して，正しく転送できるように〔LDIR〕と〔LDDR〕を使い分ける．

```
MVE  PUSH  HL        ; HL を退避
     AND   A         ; キャリをゼロに
     SBC   HL,DE     ; キャリを変化
     POP   HL        ; HL を元にもどす
     JR    NC,IRR    ; 上から転送
     DEC   BC        ; BC－1
     ADD   HL,BC     ; HL を変更
     EX    DE,HL
     ADD   HL,BC     ; DE を変更
     EX    DE,HL
     INC   BC        ; BC＋1
     LDDR            ; 下から転送
     RET
IRR  LDIR            ; 上から転送
     RET
```



53 ジャンプ命令

メモリに入れられたプログラムはゼロ番地から順次実行しますが，順序を入れ替えたり，条件によって別の番地のプログラムへ分岐したりするときに使います．

ジャンプ〔JP〕命令は，条件を伴わない無条件ジャンプと，判定条件付ジャンプ命令があり，飛び先番地は絶対番地がつけられます．

判定のジャンプ命令は条件付ジャンプと呼ばれ，フラグレジスタの内容によって，ジャンプするかジャンプせずに次の命令へいくかの決定をします．この条件には，ゼロフラグ，キャリフラグ，パリティフラグ，サインフラグがそれぞれ"0"である，または"1"であるとき，ジャンプせよというものです．条件に合わないときは，そのジャンプ命令がないのと同様に次の命令を実行します．

ジャンプ リラティブ〔JR〕命令は，ジャンプ先のアドレスを絶対値で持つのではなく，現在のプログラムカウンタ（自分自身のアドレス）からのへだたりで持っています．数値としては－128から＋127バイトで，0のときは次の命令の先頭番地です．アセンブリ言語で書くときは，飛び先番地のラベルを指定すれば，自動的に計算してくれます．ジャンプ リラティブは2バイト命令ですが，ジャンプ命令（3バイト）より実行時間がかかります．また条件判定は，ゼロフラグとキャリフラグについてのみしかできません．しかしジャンプ先までのへだたりが変わらなければジャンプ先がどこの番地に変ってもマシン語は同じになりますので，小さなプログラムモジュール内で使った場合，モジュールの配置は自由（リロケータブル）となります．

ディクリメントジャンプノンゼロ〔DJNZ〕命令は，変った命令で，Bレジスタから1を引き，ゼロでなければジャンプリラティブと同じ方式で示された番地へジャンプするというものです．繰り返しに入る前にBレジスタへ必要回数を書き，繰り返し処理の終りにこの命令をおけば，終了しなければもう1回繰り返しの先頭へジャンプ，終了すればぬけ出て次へ，と分岐させることができます．回数はBレジスタが8ビットですから256回（Bレジスタにゼロを入れておくと256回で終る）までです．

無条件ジャンプ

命令	オペランド
JP nn	飛び先のメモリにつけられた番地をそのままオペランドにつける（**絶対番地**） （例） JP 0258H ⟹ マシン語では C3 58 02（58 下位、02 上位） アセンブリ言語ではラベル名を書いてもよい オペランドには（HL）（IX）（IY）を使用してもよい．
JR nn	飛び先のメモリの番地までのディスプレイスメントをオペランドにつける（**相対番地**） （例） JR LO2 ⟹ LO2 が 0255H 番地で次の命令が 0250H 番地 とすると，マシン語では 18 05

条件ジャンプ　　条件が合えばジャンプする．合わなければ次の命令へ

命令		条件
JP	JR	
JP Z, nn	JR Z, nn	Z フラグを調べゼロの状態（Z フラグ＝1）ならジャンプ
JP NZ, nn	JR NZ, nn	Z フラグを調べノンゼロの状態（Z フラグ＝0）ならジャンプ
JP C, nn	JR C, nn	C フラグを調べキャリ有り（C フラグ＝1）ならジャンプ
JP NC, nn	JR NC, nn	C フラグを調べキャリなし（C フラグ＝0）ならジャンプ
JP PE, nn	—	P/V フラグを調べイーブン（偶数）（P/V フラグ＝1）ならジャンプ
JP PO, nn	—	P/V フラグを調べオッド（奇数）（P/V フラグ＝0）ならジャンプ
JP M, nn	—	S フラグを調べマイナス（S フラグ＝1）ならジャンプ
JP P, nn	—	S フラグを調べプラス（S フラグ＝0）ならジャンプ
—	DJNZ nn	B←B－1　B≠0 ならジャンプ

注　JR のオペランドはアセンブラによってはラベル名ではなく，相対値を書くものもある．

```
        ～
        LD    B,10H
        LD    HL,2000H
        LD    A,00H
LO1     ADD   A,(HL)
        INC   HL
        DJNZ  LO1      ──→ Bレジスタから1を減じゼロなら次の命令を実行する．
                                ゼロでなければLO1番地へ飛ぶ
        JP    KEYR     ──→ KEYRとラベルをつけた番地へ飛ぶ
                            （この次はKEYR番地の命令を実行する）
CRTR    LD    A,30H    ←── ここへは他にあるジャンプ命令等で飛んでくる．
        LD    C,00H
        ～
        JR    LO2      ──→ （LO2のラベルがつけられた命令の先頭番地－次の命令の先
                            頭番地）の値をマシン語のオペランドに持つジャンプ命令（2
KEYR    IN    A,(08H)       バイト構成の命令）．LO2とラベルをつけた番地へ飛ぶ
        CP    0
        JP    Z,KEYO   ──→ 前の命令でAレジスタと0を比較してゼロ（一致）ならKEYO
                            番地へ飛び
        CP    1        ←── ゼロでなければこの命令を実行する
        ～
KEYO    LD    B,20H
LO2     LD    HL,0038H
        ～
```

よいプログラムとは

1. 実行速度が速いこと

用途によっては，速度が問題にならない場合もあるが，まずほとんどの場合，全システムの効率に大きく影響する．

2. メモリをくわないこと

プログラム自体の長さは，無駄を省いた最少限でなければならない．
実行速度を上げるためにも重要である．また，専用システムではメモリ数を減らし，部品点数を引き下げることにより得られるメリットは多い．ただし，以下の項目を犠牲にしないよう心がけなければならない．

データ格納用メモリは，情報処理（事務計算）のような用途で多用されるが，フロッピディスク等の外部記憶を有効に利用し，システム効率を上げるような設計が望ましい．

3. デバッグ（修正）しやすいこと

いかに有能なプログラマでもでき上ったプログラムが1回で動作するのは希である．また，でき上ってから何年もたってからバグ（誤り）が発見されることがある．すみやかに対処できるよう配慮しなければならない．次項とも共通するが，コメントを豊富に，機能をモジュール化し分散させること，フラグの使用は最少限に，サブルーチンからジャンプでメインルーチンへもどるなど論外である．

4. 改造しやすいこと

新しくプログラムを作るとき，前に作ったプログラムの改造で対応できれば，こんな楽なことはない．設計変更されそうな点を予測すること．

5. 速く作ること

プログラムの原価は，ほとんど人件費につきる．速く，しかも後で応用のきくプログラムを作ることが，プログラマの使命である．

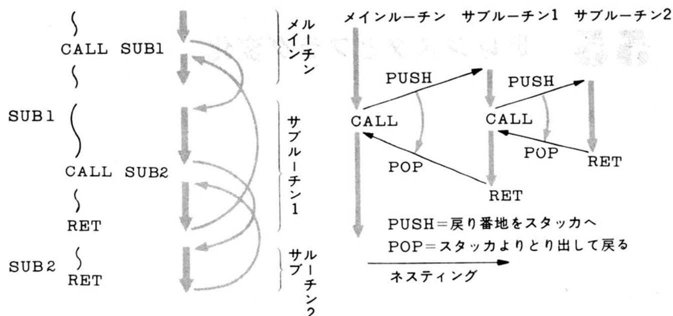
54 コール，リスタート，リターン命令(サブルーチン)

コール〔CALL〕**命令**は，ジャンプ命令と似ていますが，ジャンプするとき次の命令の番地をスタッカへ自動的にプッシュ〔PUSH〕します．そしてリターン〔RET〕命令があるとスタッカからポップ〔POP〕した値をプログラムカウンタへ入れて元の流れへ戻ります．コールで呼び出される番地からリターン命令までを**サブルーチン**と呼び，プログラムの中のどこからでも呼び出して実行させることができます．コール命令は2バイトのジャンプ先の絶対番地を持った3バイト命令です．コール命令にもジャンプ命令と同じ条件を付けることができます．条件を満たしてないときは，コール命令がないのと同様に，ジャンプせずに次の命令を実行します．

リスタート〔RST〕**命令**は，特定の8個の番地へのコール命令に相当します．1バイト命令なので，多用されるサブルーチンをコールするときに便利な命令です．

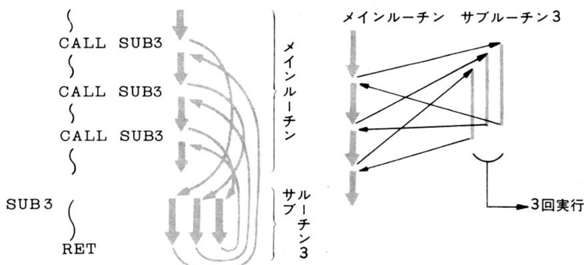
リターン命令であってもノン マスカブル 割り込み〔NMI〕からのリターンは〔RETN〕を使います．他の割り込みに対する割り込み禁止状態を解除するためです．また，割り込み（INT）からのリターンは，Z-80のペリフェラルのデージーチェーンによる優先順位決定機能を使うときは，リターン フロム インタラプト〔RETI〕命令を使って，ペリフェラルに対して割り込み処理の完了を知らせてやる必要があります．これは割り込み処理ルーチンの中でサブルーチンコールを行なう場合もあり，サブルーチンからのリターンで割り込み処理完了になってしまわないようにするためです．

サブルーチンの使い方は，プログラムの良否にかかわる重要なテクニックです．単に同じ手続きをあちこちで何回も使うから，記述の手間を省くというだけでなく，プログラムの機能のブロック化，あるいは単機能モジュールのブラックボックス化に役立ちます．デバッグ済のサブルーチンは内容を知らなくても使い方だけを知っておけば，いつでも呼び出せます．モジュールとしてたくさん作っておけば，次からはモジュールの組み合せを変えるだけで新しいプログラムができ上るわけです．



（例1）メインルーチンからサブルーチン1をコールし，さらにサブルーチン2をコールする

（例2）メインルーチンのあちこちから1つのサブルーチン3をコールする



命　令	オ　ペ　ラ　ン　ド	
CALL nn	nnは絶対番地	無条件コール
CALL cc，nn	nnは絶対番地 ccは条件	条件付コール　前項のJPと同じ条件
RET	オペランドはない	無条件リターン
RET cc	ccは条件	条件付リターン　前項のJPと同じ条件
RST n	nは 00H 08H 10H 18H 20H 28H 30H 38H のいずれかを選ぶ n=08Hとすると，0008Hへのコール命令（CALL 0008H） と同じ働きをする．ただしRST nは　1バイト命令	

55 Fレジスタとフラグ変化

Fレジスタは六つのフラグにより構成されます．フラグはそれぞれが意味を持ち，演算命令やローテート命令，入出力命令などの実行結果によって変化します．

キャリ フラグ（Cフラグ）は，最上位のビットからの桁上り，桁がりによりあるいはローテート，シフトで変化します．

ゼロ フラグ（Zフラグ）は，実行結果がゼロになったとき"1"にセットされます．

パリティ/オーバフローフラグ（P/V フラグ）は，パリティ（ビットパターンの"1"の数が奇数か偶数か）と，符号付演算のオーバフローを兼ねています．パリティは，奇数なら"0"，偶数なら"1"，オーバフローは，オーバフローして本来正の数になるべき結果が負の数になってしまったとき"1"，正しい結果のとき"0"になります．

サイン フラグ（Sフラグ）は，符号付演算の符号ビット（左端ビット）と同じになります．負数であれば"1"，正数であれば"0"です．

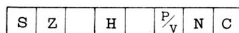
ハーフ キャリ（Hフラグ）は，下位4ビットに対するキャリ フラグです．

これらのフラグのうち，サブトラクト フラグとハーフ キャリフラグは，ジャンプ命令では判定できませんが，2進化10進数を扱うデシマル アジャスト アキュムレータ〔DAA〕命令を実行するとき参照されます．

フラグは命令により変化する，変化しない，不定になる，各場合があります．

コンプリメント キャリ フラグ〔CCF〕命令は，他のレジスタに変化を与えずにキャリフラグのビットパターンを反転させる命令です．セット キャリ フラグ〔SCF〕命令はキャリフラグを"1"にする命令です．なお"0"にしたいときは〔AND A〕か〔OR A〕を使います．

Fレジスタ



キャリフラグ
加算，減算で桁上り，桁がりがあったとき1になる．
ローテート，シフトでは，各ビットにつれて移動する．

サブトラクトフラグ
（DAA命令実行用＝プログラムでは操作できない）
命令によりセットされるかリセットされるか決まっている．
加算系のとき0，減算系のとき1になる．

P/V フラグ
命令によりP：パリティフラグとV：オーバフローフラグに使いわける．

パリティフラグ
論理演算，Cレジスタ間接ポート指定の入力命令等で，ディスティネーションに乗ったデータのビットパターンの1の数が偶数（イーブン）のとき1，奇数（オッド）のとき0になる．

オーバフローフラグ
演算の結果が符号付算術演算の結果としてオーバフローしたとき1．すなわち8ビット演算では－128～＋127，16ビット演算では－32768～＋32767以外を表現する必要が出たときセットされる．したがって，正数のみを対象とした符号なし演算，(8ビットでは0～255，16ビットでは0～65535の範囲で定義した演算)のときは無視してよい．この場合はCフラグがオーバフローを表わしている．

このビットは使用していない

ハーフキャリフラグ
（DAA命令実行用＝プログラムでは操作できない）
8ビットの演算で下4ビットからの桁上げ，桁下がりがあったとき1になる．

このビットは使用していない．

ゼロフラグ
実行の結果がゼロのとき1，ゼロ以外のとき0になる．

サインフラグ
実行の結果が符号付算術演算の結果として負の数になったとき（この場合最上位ビットが1になる）1になる．

56 2進化10進数と10進補正命令

CPU内の算術演算は，メモリやレジスタに記憶されたビットパターンを2進数として演算します．8ビットレジスタでは0〜255まで，256以上を表わすと桁上げをしますが，レジスタの内容は0と256が同じパターンになります．多くの桁を扱う場合は，桁上げ操作や入出力を簡単にするため，2進化10進数（バイナリー コーデッド デシマル＝BCD）を用います．

4ビットのパターンは16種類0〜Fまでありますが，このうち0〜9までを有効とし，A〜Fについては上位桁の4ビットへ1繰上げ0〜5にします．このようにすると4ビット＝1桁の10進数表現ができます．10桁の10進数を使いたいときはメモリに5バイトのエリアを確保し，2桁ずつ演算するのです．

しかし，演算はBCDで表現されていても，2進数で計算しますので，結果としてA〜Fが出現してしまいます．ここでデシマル アジャスト アキュムレータ〔DAA〕命令を実行すると，いまの計算が加算か減算か，ハーフキャリは出ているか，などフラグレジスタの状態によって必要な桁上げ，桁がり処理をして，BCDに戻してくれます．BCDはデータエリアが多少大きくなりますが，入出力はほとんどの場合10進数が要求されますので，変換の手間が省け，多桁演算がやりやすいなど特徴が多く，よく使われる手法です．

ビットパターン	名前	BCD 表現
0 0 0 0	0	ここまでを 10進数に対応させる 1010は 上の位へ桁上げし 0000に戻る
0 0 0 1	1	
0 0 1 0	2	
0 0 1 1	3	
0 1 0 0	4	
0 1 0 1	5	
0 1 1 0	6	
0 1 1 1	7	
1 0 0 0	8	
1 0 0 1	9	
1 0 1 0	A	使わない
1 0 1 1	B	
1 1 0 0	C	
1 1 0 1	D	
1 1 1 0	E	
1 1 1 1	F	

8＋6 を例にとって
加算すると
10進数では
8＋6＝14 であるが
2桁のBCDでは

```
   0000 1000 = 08
+) 0000 0110 = 06
   0000 1110 = 0E
```

となり，下4ビットは使わない値に
なってしまう．
ここでDAA命令を実行すると

0000 1110



0001 0100
⇩　⇩
1　4

になおしてくれる
この補正のし方は加減算，
桁あふれ等により異なるが，
Fレジスタの働きで間違えることはない

BCDの表し方

1001 0110
9　6
↑　↑
10の位　1の位
2桁の10進数

(4ビットを1デジット（桁）と呼ぶのは
ここからきている)

2進数

1001 0110 は
16進数 になおすと（名前で呼ぶと）
96 である．これを
10進数になおすと
$9\times16^1+6\times16^0=150$ である

これをBCDと定義すると

1001 0110 は
16進読み にすると
96（きゅうろく）
10進数にしても
96（きゅうじゅうろく）である

コンピュータの中で数値を表わすには

1. 2進数として扱う
2. BCDとして扱う
3. 指数表現をとり入れる

等のやり方がある

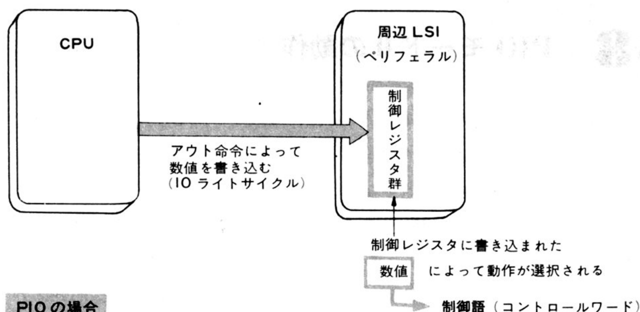
1．は入出力のたびに10進数との変換が必要．プログラムは簡単でスピードも早い．桁数が自由にならない．
2．は桁数が自由になり，入出力，プログラムは簡単．メモリにむだが多い．
3．は，桁数が多い場合に有効．プログラムは複雑で演算時間がかかる．

57 ペリフェラルのプログラミング

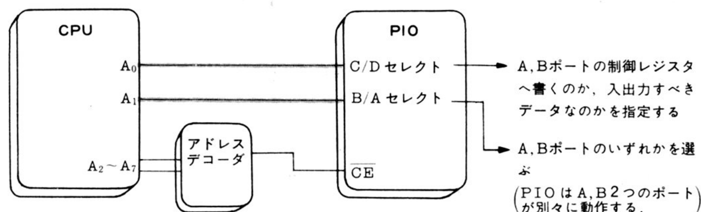
ペリフェラルを使うときは，データの入出力を行なう前に，ペリフェラルの内部レジスタへ情報を書き込んで，目的にあった動作をするように設定しなければなりません．機能が多いほど選択の幅が広くなりますので，たくさんの情報を与えなければなりません．情報は次々に書き込みます．情報の中に何の情報かを示すビットが設けられている場合もありますが，書き込む順序によって，意味が違ってしまうことがあります．マニュアルに記載された順序を守れば安全です．

ペリフェラルのレジスタへ情報を書き込むには，そのペリフェラルのつながっているIOポートへ出力命令を出します．PIOとSIOには，C/Dセレクト信号端子があり，"1"のときCすなわちコントロールワード，"0"のときDすなわちデータ，とセレクトしますので，制御情報を書き込むときはこの端子を"1"にして出力命令を出します．実際にはこの端子をアドレスバスのいずれかのビットへつなぎますと，制御ワードとデータとは別のIOポートアドレスに配置されることになります．CTCのチャンネルセレクトやPIOのB/Aポートセレクトも同じ考え方で対処します．

一度書き込んだ制御情報は，リセットされるか，または新たに書き込まれるまで有効です．また，特別な場合を除いてIN命令で読み出すことはできません．



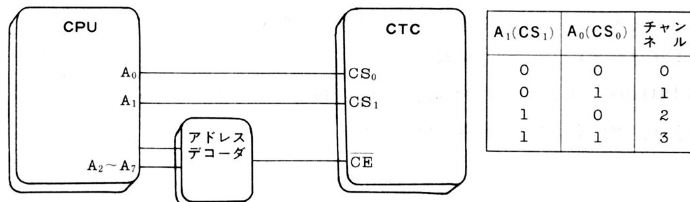
PIOの場合



$A_7 \sim A_2$	A_1	A_0	番　　地	ポート切換え	データ/制御語切換え
×	0	0	×0，×4，×8，×Cのどれか	B/Aセレクトが"0"だから	C/Dセレクトが"0"だから**データ**
×	0	1	×1，×5，×9，×Dのどれか	**A** ポート	〃　"1"　〃　**制御語**
×	1	0	×2，×6，×A，×Eのどれか	B/Aセレクトが"1"だから	〃　"0"　〃　**データ**
×	1	1	×3，×7，×B，×Fのどれか	**B** ポート	〃　"1"　〃　**制御語**

×はアドレスデコーダによってきまる

CTCの場合　(CTCに対する入出力は制御レジスタにのみ適応する
CS_0 CS_1 は内部の4つのチャンネルを選択する)



$A_1(CS_1)$	$A_0(CS_0)$	チャンネル
0	0	0
0	1	1
1	0	2
1	1	3

58 PIO モード 0 の動作

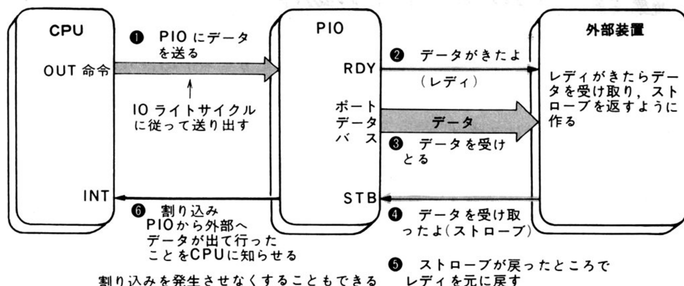
モード0は出力モードです．8ビットの入出力線に出力〔OUT〕命令で出されたデータバスの状態がそのまま出てきます．データバスの内容が変っても次に出力命令がくるまで出力線の内容はそのままです．

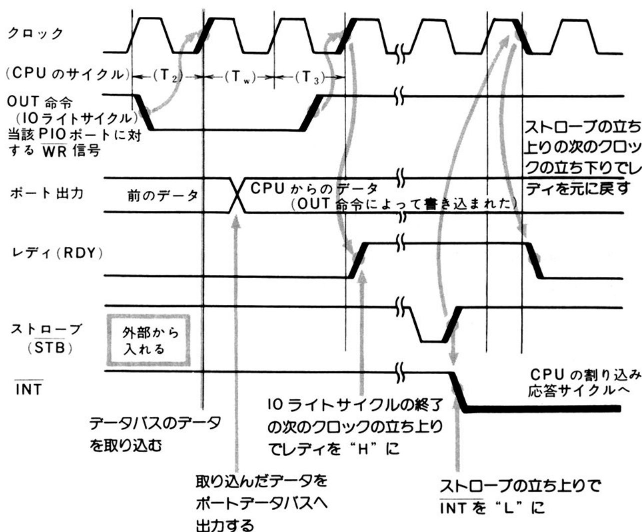
CPUからの出力動作では，IOリクエスト（$\overline{\text{IORQ}}$）とライト（$\overline{\text{WR}}$）信号が"L"になりますが，PIOにはライト信号の入力端子がありません．これは（$\overline{\text{IORQ}}$）が"L"で，リード（$\overline{\text{RD}}$）が"H"で，チップイネーブル（$\overline{\text{CE}}$）が"L"で，$\overline{\text{C/D}}$がデータすなわち"L"のときPIO内部でライト信号を作り出しているからです．

出力命令が出されるとデータバスからPIOの出力レジスタへ信号が入ります．内部のライト信号が立ち上ると次のクロックの立ち下りで，すでに出力線に出力レジスタの内容が乗り確定しています．同時にレディ端子からレディ（RDY）信号が出されます．外部装置はこのレディ信号が"H"になったことにより出力線のデータを取り込み，終ったらストローブ（$\overline{\text{STB}}$）端子を"L"にして応答します．レディ信号は，このストローブの立ち上りの次のクロックの立ち下りで"L"に戻り，次のデータの出力命令を待ちます．割り込みがイネーブルにプログラムされていて，優先順位の高い割り込みがかかっていなければ，ストローブの立ち上りで割り込み（$\overline{\text{INT}}$）出力は"L"になってCPUに割り込みをかけることができます．次の出力命令があるまで出力線は変わりません．

レディ信号が"H"のとき，すなわち出力命令に対してストローブが返っていないときにさらに出力命令を出すと，レディ信号は一度"L"に戻ってすぐ"H"になります．レディ信号は正論理信号ですから"H"アクティブです．

レディとストローブの信号線をハンドシェーク線と呼びます．外部装置とCPUはPIOのハンドシェーク線を通じて互いに確認し合いながら（同期をとりながら）データのやりとりを行なうわけです．


CPU
OUT 命令
INT
❶ PIO にデータを送る
IO ライトサイクルに従って送り出す
❻ 割り込み
PIOから外部へデータが出て行ったことをCPUに知らせる
割り込みを発生させなくすることもできる
PIO
RDY
ポートデータバス
STB
❷ データがきたよ
（レディ）
データ
❸ データを受けとる
❹ データを受け取ったよ（ストローブ）
❺ ストローブが戻ったところでレディを元に戻す
外部装置
レディがきたらデータを受け取り，ストローブを返すように作る

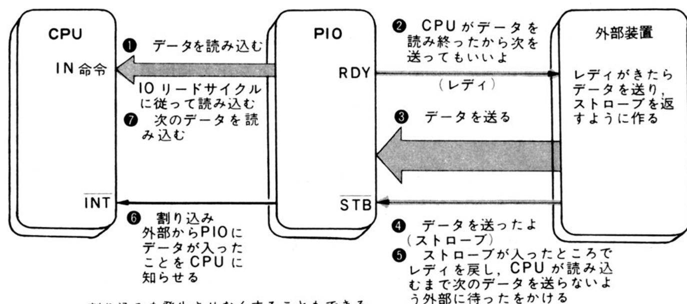

クロック
（CPU のサイクル）
（T2）
（Tw）
（T3）
OUT 命令（IOライトサイクル）当該PIOポートに対するWR信号
ポート出力
前のデータ
CPU からのデータ（OUT 命令によって書き込まれた）
レディ（RDY）
ストローブ（STB）
外部から入れる
INT
ストローブの立ち上りの次のクロックの立ち下りでレディを元に戻す
CPU の割り込み応答サイクルへ
データバスのデータを取り込む
取り込んだデータをポートデータバスへ出力する
IO ライトサイクルの終了の次のクロックの立ち上りでレディを"H"に
ストローブの立ち上りでINTを"L"に

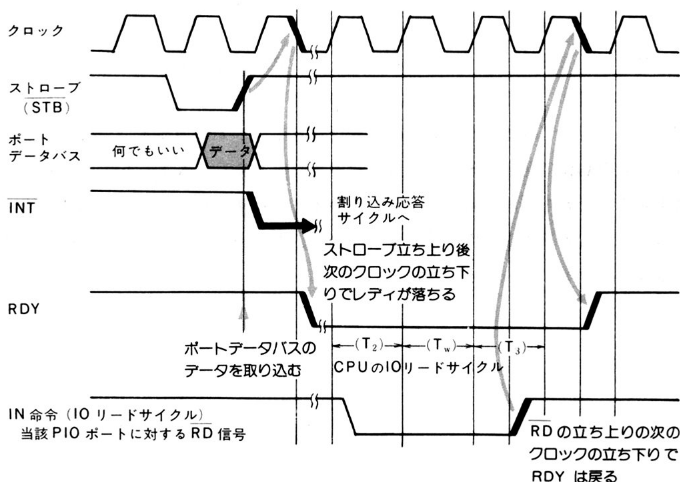
59 PIOモード1の動作

モード1は入力モードです．外部から入ってきた8ビットの信号がポートの入出力線に与えられ，CPUからの入力〔IN〕命令でデータバスを通じてCPUの内部レジスタへ取り込みます．

レディ端子は，リセットがかかると"L"になっています．CPUからの入力命令で"H"になって受け入れ可能を外部へ知らせます．この一発目の入力命令で読み込んだデータは無意味です．外部装置からレディが"H"のとき，入出力線にデータを与え，続いてストローブを"L"にするとストローブの立ち上りでPIOは入出力線から内部の入力レジスタへデータを取り込みます．次のクロックの立ち下りでレディを"L"にして外部から次のデータが入ることを禁止し，同時に割り込み可の状態であれば，CPUに割り込みをかけます．割り込み処理ルーチンで，入力〔IN〕命令を実行し，このデータをCPUの内部レジスタへ読み込みますと，リード（$\overline{RD}$）信号の立ち上りと次のクロックの立ち下りでレディを"H"に戻し，次のデータ入力に備えます．

モード1では1個のダミーの入力命令が必要です．プログラミングが終り外部からの信号を受け入れてもよい状態になったら〔IN〕命令を実行してレディ信号を出し外部装置に受け入れ可を知らせて下さい．レディ信号が受け入れ状態を示していないときに，入力線にデータを乗せてストローブ信号を与えても，入力レジスタ上へデータを書き込むことはできます．しかし何らかの方法でCPUの入力命令の実行が遅れないようにしないと，データが失われるおそれがあります．


CPU
IN命令
❶ データを読み込む
IOリードサイクルに従って読み込む
❼ 次のデータを読み込む
INT
PIO
RDY
STB
❷ CPUがデータを読み終ったから次を送ってもいいよ
(レディ)
❸ データを送る
外部装置
レディがきたらデータを送り，ストローブを返すように作る
❻ 割り込み
外部からPIOにデータが入ったことをCPUに知らせる
割り込みを発生させなくすることもできる
❹ データを送ったよ
(ストローブ)
❺ ストローブが入ったところでレディを戻し，CPUが読み込むまで次のデータを送らないよう外部に待ったをかける

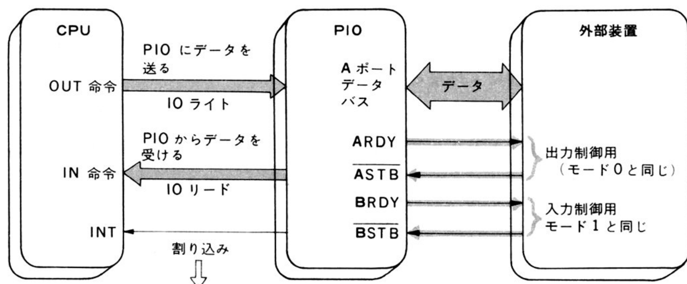

クロック
ストローブ
(STB)
ポート
データバス
何でもいい
データ
INT
割り込み応答
サイクルへ
ストローブ立ち上り後
次のクロックの立ち下りでレディが落ちる
RDY
ポートデータバスのデータを取り込む
(T2)
(Tw)
(T3)
CPUのIOリードサイクル
IN命令(IOリードサイクル)
当該PIOポートに対するRD信号
RDの立ち上りの次のクロックの立ち下りでRDYは戻る

60 PIO モード2の動作

モード2はAポートだけについて可能な，入出力モードです．8本のデータ線は入出力共用になります．ハンドシェーク線はAポートが出力制御用，Bポートが入力制御用になります．したがってBポートはハンドシェーク線を必要としないモード3に設定しなければなりません．

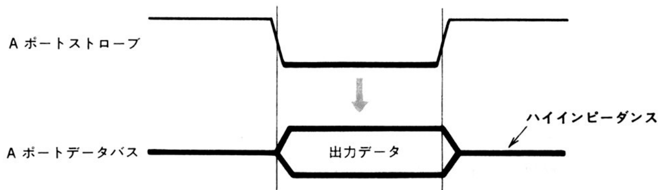
出力命令によるデータ出力動作はモード0と同じですが，モード0では出力線に常にデータが乗せられているのに対し，モード2のときはAポートのストローブ（$\overline{\text{ASTB}}$）が"L"になっている間だけ，多少遅れて乗せられてきます．外部装置はこのストローブを立ち上げるタイミングでデータを読み取ればよいのです．入力命令によるデータ入力動作は，Bポートのハンドシェーク線を使うほかはモード1と同じです．

割り込みをかけるタイミングも，モード0，モード1と同じですが，入力動作ではBポートのハンドシェーク線を使うため，Bポートに書き込まれたベクトルが送出されます．したがって入力，出力各動作での割り込みを別々のベクトルで制御することができますが，Bポートをビットモードで使用する場合は，本来のBポートの動作で発生する割り込みに対するベクトルと，Aポートの入力動作で発生するベクトルが同じ値になってしまいます．いずれかの割り込み機能を使用しないようにするか，処理プログラムの中で，どちらの割り込みかを判断しなければなりません．



入力動作と出力動作ではベクトルが異なる → 異なるように設定しておくこと

Bポートデータバスはモード3で使う．または使わない．ただしモード3に設定しておくこと



Aポートストローブが“L”のときだけ出力データがポートデータバスに現われる．
したがってこのときは外部からデータを与えてはならない．
他は，モード0，モード1と同じ．

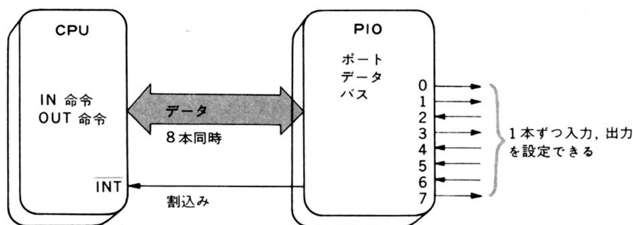
61 PIOモード3の動作

モード3はハンドシェーク線を使用しないで，非同期で入力出力を行ないます．ビット単位に入出力の設定ができ，割り込みは入力に指定したビットが指定した状態になったときに入ります．

出力命令で送られたデータはモード0と同じタイミングで出力に指定されたビットに乗せられます．入力命令で読み込むと，リード($\overline{RD}$)信号が立ち下る直前の入力に指定されたビットの状態がCPUに読み込まれます．出力に指定されたビットはそれ以前に出力命令で送り出したままが読み込まれます．

入力に指定したビットは，さらに割り込みに関係させるか否かのマスク指定をすることができます．マスク指定で割り込み要因のモニタビットに指定したビットが全部そろったとき（AND条件），いずれか一つが入ったとき（OR条件)を選択でき，入力線を正論理（"H"アクティブ）としてとらえるか負論理（"L"アクティブ）としてとらえるかの選択もできます。

コントローラとしての用途では，PIOをモード3で使うことが多いと考えられます．周期的にCPUは入力ビットの状態を検査し，対応した出力信号を出力ビットへ乗せるようなプログラムを組みますが，入力ビットを読み込む周期はプログラムの長さで決まります．もっと早い対応が必要なシステムでは，割り込みを使うことにより解決できます．ただし割り込みによるシステムの応答は，外部装置で起こり得る最悪の条件のときを想定して設計しないと，CPUが追いつかなくなることもあります．時間計算から許容ステップ数を出し，その範囲内のプログラムを作るのですが，システムクロック周波数の決定とも合わせて慎重に検討しなければなりません．



割り込み発生の条件，設定

1．入力ビットのうちどれを判断の対象とするか（マスク）
2．入力は“H”アクティブか“L”アクティブか　（論理）
3．AND か OR か　（AND/OR）

たとえば，上の例で

・ビット4と6が共にアクティブになったとき割り込みを発生させたい
・入力は普通は“H”になって信号があるときだけ“L”になる（“L”アクティブ）

とすると

・マスク指定はビット4と6
・論理は負論理
・AND/OR は AND　と指定する

具体的には（次項参照）

ベクトル	××××××× 0	⇨	××
モード設定	1 1 0 0 1 1 1 1	⇨	C F
ビット指定	0 1 1 1 0 1 0 0（←7ビット目 … ←0ビット目）	⇨	7 4
割込み制御語	1 1 0 1 0 1 1 1	⇨	D 7
マスク指定	1 0 1 0 1 1 1 1	⇨	A F

⇩

これを順次PIOの
制御レジスタ群へ
OUTする

62 PIOのプログラミング

割り込みベクトル

7	6	5	4	3	2	1	0

0 → "0" であることにより割り込みベクトルを意味する

7〜0 → 割り込み処理ルーチンのアドレステーブルの番地（下位）

モード設定

7	6	5	4	3	2	1	0

3〜0 → "1111" であることによってモード設定語を意味する

5, 4 → 無意味（何でもよい）

7, 6 → モード

7	6

00＝モード0（出力モード）
01＝モード1（入力モード）
10＝モード2（双方向モード）
11＝モード3（ビットモード）

ビット指定　モード設定でモード3を指定したときは続けてこの指定をします．

7	6	5	4	3	2	1	0

7〜0 → ポートの入出力線のビット位置に対応したビットを，
入力ビットにするとき "1"
出力ビットにするとき "0" に指定する

> 制御語の書き込みはAポート，Bポートを別々に行なって下さい．
> A，B混在させると，動作しない場合があります．順序もここに書かれているとおりなら問題ありません．
> 割り込みを使わない場合は，ベクトルの書き込みは省略できます．

割り込み制御語

7	6	5	4	3	2	1	0

- ビット3〜0 → "0111"であることにより割り込み制御語を意味する
- ビット4（マスク）→ モード3で割り込みに関係する入力ビットがある場合，続けてマスク指定を行なう．このとき"1"
  割り込みを使わないとき，マスク指定はしなくてよいので，このときは"0"
- ビット5（H/L）→ モード3で割り込み発生要因となる入力ビットが"H"アクティブ(正論理)なら"1"，"L"アクティブ(負論理)なら"0"
- ビット6（AND/OR）→ モード3で，割り込み発生要因となる入力ビットのすべてがアクティブなときに割り込み発生する（AND）なら"1"
  いずれか一つ以上アクティブなとき発生(OR）なら"0"
- ビット7（割り込み）→ 割り込みを発生させるとき"1"，発生させないとき"0"

マスク指定

7	6	5	4	3	2	1	0

- ビット7〜0 → 割り込み制御語のビット4が"1"のときは続けてこのマスク指定をする．ポートの入力線のビット位置に対応するビットを"1"にすると，割り込み発生とは無関係な入力になり，"0"にしたビットのみが割り込み発生要因となる

割り込み制御語

7	6	5	4	3	2	1	0

- ビット3〜0 → "0011"であるとき，割り込み発生の有無のみを変更する制御語を意味する
- ビット6〜4 → 無意味（以前の指定を変えない）
- ビット7 → 割り込みを発生させるとき"1"，発生させないとき"0"

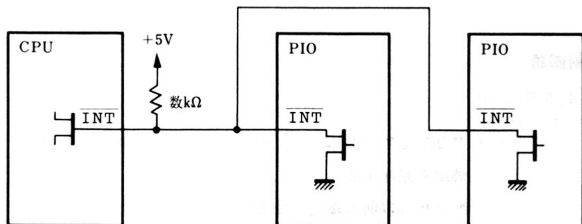
63 PIOのプログラム例

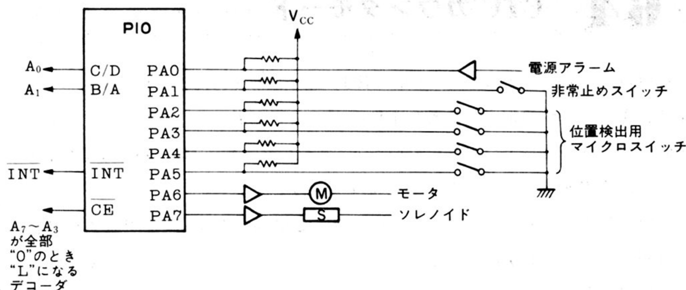
PIOをモード3で使用する場合の一例を考えてみます．PIOはIOポートの00H～03Hに配置されているものとします．Aポートの0～5ビットを入力とし，他を出力ビットにします．電源異常信号と非常停止信号が共に"L"アクティブで0と1ビットにつながっています．処理に速度が要求されますので，割り込みをモード2で使用します．他の入力は位置検出のマイクロスイッチの信号で出力はすべてモータやソレノイドを駆動します．出力を"H"にすると動き，"L"にすると止まります．応答速度は問題にならないとします．

まず0000H番地からのルーチンでPIOを設定します．設定の最後でCPUの割り込み受付を可能にします．次に入力ビットを読み込み，その条件に従った制御情報をAレジスタにととのえ出力ビットへ出力します．

割り込みが入ると，すべての出力を"L"にして動作を止めます．CPUはホルトに入り，新たにリセットがかかるのを待ちます．

PIOに限らず，ペリフェラルの$\overline{\text{INT}}$の出力端子はオープンドレインになっています．したがって，＋5Vへ抵抗でプルアップしておかなければなりません（ワイアードオア）．






```
; CONTROL                     コメント行
      ORG   0000H    ; このプログラムをゼロ番地から配置
      LD    SP,0000H ; スタックポインタイニシャライズ
      IM    2        ; 割り込みモード        ┐
      LD    A,01H    ; I レジスタセット      │
      LD    I,A                              │
      LD    A,00H    ; ベクトル              │
      OUT   (01H),A                          │
      LD    A,0CFH   ; PIO モード設定        ├ PIO のプログラミング
      OUT   (01H),A                          │
      LD    A,3FH    ; ビット指定            │
      OUT   (01H),A                          │
      LD    A,97H    ; 割り込み制御語        │
      OUT   (01H),A                          │
      LD    A,0FCH   ; マスク指定            ┘
      OUT   (01H),A
      EI             ; 割り込み許可
JOBS  ∫              ; 仕事を始める

      IN    A,(00H)  ; 入力ビットを読む
      ∫
      OUT   (00H),A  ; 出力をコントロールする
      JP    JOBS     ; 次の仕事を始める

      ORG   0100H
      DEFW  INTR     ; 割り込み処理ルーチンの先頭番地定義
;                    ; 割り込み処理ルーチン
INTR  LD    A,0H     ; モータを止める
      OUT   (00H),A
      HALT           ; CPU を止める．リセット信号が入るまで停止
      END            ; プログラム終り
```

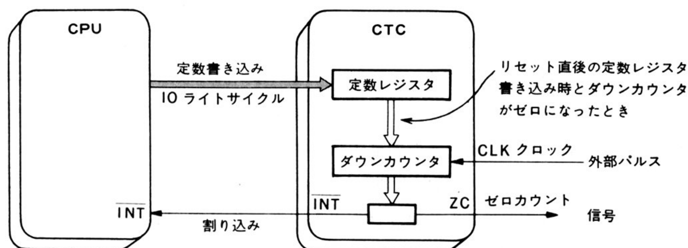
64 CTCカウンタモード

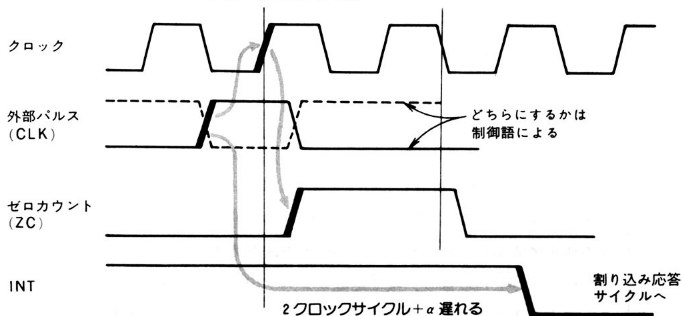
CTCのクロック/トリガ（CLK/TRG）端子に与えられるパルスをカウントします．あらかじめ設定された数を1個のパルスで1ずつ減らしてゼロになると，次のクロックの立ち上りでゼロカウント/タイムアウト（ZC/TO）端子を1.5クロックの間"H"にし，割り込みを発生します．

カウントするパルスはシステムクロックの2倍以上の周期で，"H"と"L"の時間はおのおの120 ns以上必要です．パルスの立ち上りまたは立ち下りエッジ（指定できる）がくると，次のクロックの立ち上りでダウンカウンタを1減じます．ダウンカウンタがゼロになるとゼロカウント信号を出し割り込みをかけます．同時に定数レジスタに設定されている数値をダウンカウンタへ書き込み，次のカウントに備えます．カウント途中で定数レジスタに新しい数値を書き込むと，次のゼロカウント時から有効になります．定数レジスタからダウンカウンタへの書き込みは，リセット状態から最初に定数レジスタが設定されたときと，ダウンカウンタがゼロになったときです．定数レジスタは8ビットですから1～256の値が設定できます．0を設定したときは256を意味します．

割り込みベクトルレジスタはCTC 1個＝4チャンネル分に1個しかありません．ベクトルの設定はチャンネル0に書き込みます．割り込みが発生すると要求するチャンネルによって，ビット1とビット2が決まったパターンに修飾されて送り出されてきます．優先順位はチャンネル0が一番高く順にチャンネル3が最後位です．

制御語や定数の書き込み読み出しのときのチャンネル指定はチャンネルセレクト（CS_0）と（CS_1）を切り替えて行ないます．アドレスバスの2本（A_0とA_1がよく使われる）を接続すればCPU側からは四つのチャンネルが一連のポートアドレスに配列されます．PIOのC/DセレクトB/Aセレクトと同じ考え方です．


CPU
CTC
定数書き込み
IO ライトサイクル
定数レジスタ
リセット直後の定数レジスタ
書き込み時とダウンカウンタ
がゼロになったとき
ダウンカウンタ
CLK クロック
外部パルス
INT
INT
ZC
ゼロカウント
信号
割り込み


ゼロカウントになった場合
クロック
外部パルス
(CLK)
どちらにするかは
制御語による
ゼロカウント
(ZC)
INT
割り込み応答
サイクルへ
2クロックサイクル+α 遅れる

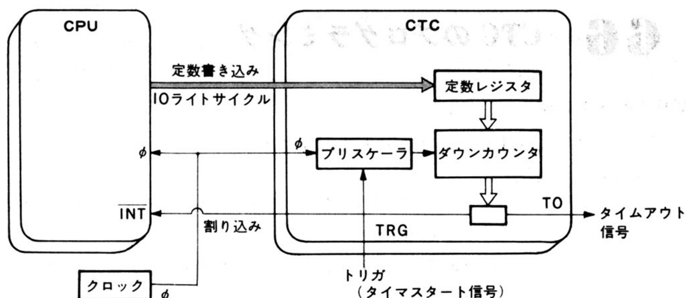
65 CTCタイマモード

カウンタモードでは，不定期的なパルスをカウントしますが，決まった周期を持ったパルスをカウントすることにより時間経過を知ることができます．これがタイマモードの動作の基本です．カウントするパルスはプリスケーラで1/16か1/256に分周されたシステムクロックを使います．システムクロックの周期と分周数と，定数レジスタに書かれた数値の積がゼロカウント/タイムアウト（ZC/TO）端子へ信号が出てくる時間周期となります．

タイマの起動は，トリガと自動が選択できます．トリガ起動の場合は，クロック/トリガ端子の立ち上りか立ち下りのエッジ（選択できる）から2回目のクロックの立ち上りでスタートします．自動の場合は，時間定数を書き込む出力命令の次の命令サイクルと同時にスタートします．

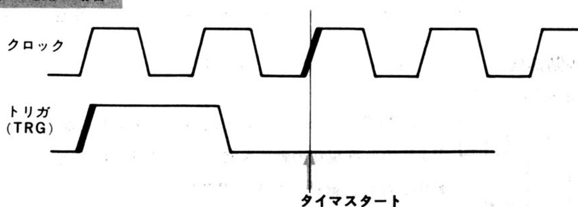
割り込みの発生と，定数の再設定についてはカウンタモードと同じです．

CTCはカウント中に読み出しても差しつかえありません．ダウンカウンタの現在値を知ることができます．ゼロカウント/タイムアウトを検知するのにZC/TO信号も割り込みも使わず，ダウンカウンタの値を読み出し，ゼロかどうか調べる方法が考えられますが，ダウンカウンタがゼロになり，次に定数レジスタの値が再びダウンカウンタへ書き込まれるまで，2クロックしかないので，この間にCPUが読み出すとは限らないため，不確実で実用できません．

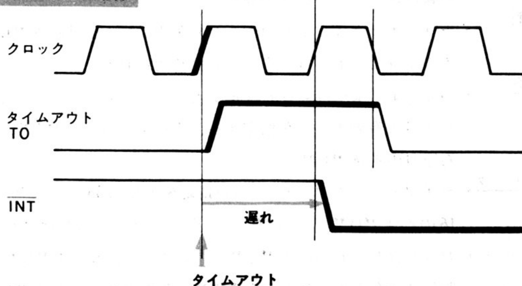


前項ではφは省略してあるが，
カウンタモードでもクロックは
与えなければならない

トリガ起動の場合



タイムアウトの場合



66 CTCのプログラミング

割り込みベクトル

（チャンネル0に対してのみ書き込むことができます）

7	6	5	4	3	2	1	0

- 0 → ここが"0"であることによって割り込みベクトルであることを意味する
- 2 1 → 何を書いてもよいが，割り込みを要求するチャンネルにより 2 1
  - 00 = チャンネル0
  - 01 = チャンネル1
  - 10 = チャンネル2
  - 11 = チャンネル3　があてはめられる
- 7〜3 → 割り込み処理ルーチンのアドレステーブルの番地（下位）

チャンネル制御語（各チャンネルごとに書き込みます）

7	6	5	4	3	2	1	0

- 0 → ここが"1"であることによって，チャンネル制御語を意味する
- 1 リセット → "1"のとき現在の動作を停止し，時間定数を書き込むまで動作しない．継続のときは"0"
- 2 定数 → 次に書き込むデータが時間定数であるとき"1"，時間定数の書き込みがないとき"0"
- 3 起動 → タイマモードで，トリガパルス起動"1"，自動起動"0"　カウンタモードではこのビットは無視される
- 4 エッジ → タイマモードではトリガパルスの，カウンタモードでは入力パルスの有効エッジを指定する．立ち上りのとき"1"，立ち下りのとき"0"
- 5 プリスケーラ → タイマモードでプリスケーラの指定をする．256のとき"1"，16のとき"0"，カウンタモードではこのビットは無視される
- 6 モード → カウンタモードのとき"1"，タイマモードのとき"0"
- 7 割り込み → 割り込みを発生させるとき"1"，発生させないとき"0"

時間定数

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

→ チャネル制御語のビット2が"1"のとき続けて書き込む．各ビットは2進数でゼロカウントになるまでの数値を表現している．ゼロのときは256として扱われる

ファンイン ファンアウト

Z-80ファミリはE-D-MOSプロセスという製法でできております．TTLのようなバイポーラとは異なり，規格の決め方にも相違があります．

入力端子は，0.8V以下で"L"，2.0V以上で"H"になり，電流は0～V_{CC} Vに対して10μA以下しか流れません．ファンインとしてはごく小さいものです．

出力端子は，"L"が1.8mA流れるとき0.4V以下，"H"が250μA流れ出して2.4V以上となっており，ファンアウトとしてはノーマルタイプのTTL 1個分，LSタイプなら4個までドライブできる能力を持っていることになります．応用機器の必要ノイズ裕度や信頼性に応じてチェックしてから使用して下さい．

参　考　文　献

シャープ：Z-80テクニカルマニアル，エレクトロニクスダイジェスト社

渡部弘之訳：コンピュータ・データ通信技術，CQ出版

森亮一監修：Z-80マイクロコンピュータ，丸善

森下　巌：マイクロコンピュータ入門，昭晃堂

大川善邦：マイコン入門，コロナ社

平松啓二，森本淳堯：マイコン入門心得帖，オーム社

平松啓二，森本淳堯：続マイコン入門心得帖，オーム社

小牧常松，大條　廣：図解マイコンの使い方，オーム社

小牧常松，大原茂之：図解マイコンのためのアセンブラ入門，オーム社

平松啓二，斎藤　剛：図解マイコンのインタフェース，オーム社

矢田光治：図解マイコンの基礎知識，オーム社

各社半導体規格表

例　　　題

プログラム　1　（ループ）

プログラム　2　（判　断）

プログラム　3　（メモリクリアサブルーチン）

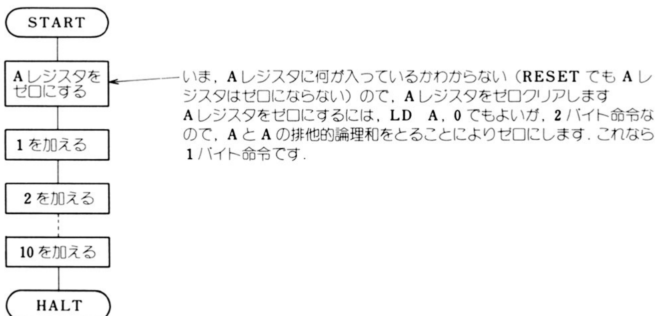
プログラム　4　（変換-テーブルサーチ）

プログラム　5　（スイッチの表示）

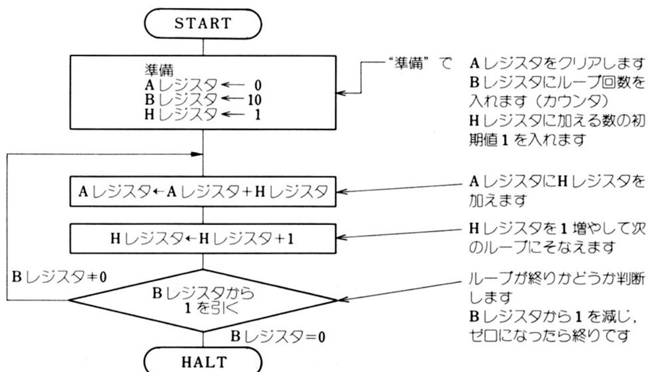
プログラム　6　（スイッチの表示-割り込み）

プログラム 1 (ループ)

1から10までの整数を加えるプログラムを考えます．答はAレジスタに入っていればよいとします．〈SAMPLE 1〉



これが1～100までだったら上のやり方では大変です．そこで「くり返えし」を使います．〈SAMPLE 2〉



この方法なら，Bレジスタに入れる数を変えればいくつでも加えられます．ただし，このままではAレジスタがオーバフローすることがありますので工夫がいります．

〔注〕 サンプルプログラムを全部一連にしてアセンブルしたものです．各サンプルプログラム間に関連はありません．実行するときにはモニタプログラムを持ったワンボードコンピュータ．たとえばシャープの SM-B-80TE を使って下さい．

行番号　ソースプログラム（キーを打ってインプットする）

プログラムを見やすくするためのコメント行

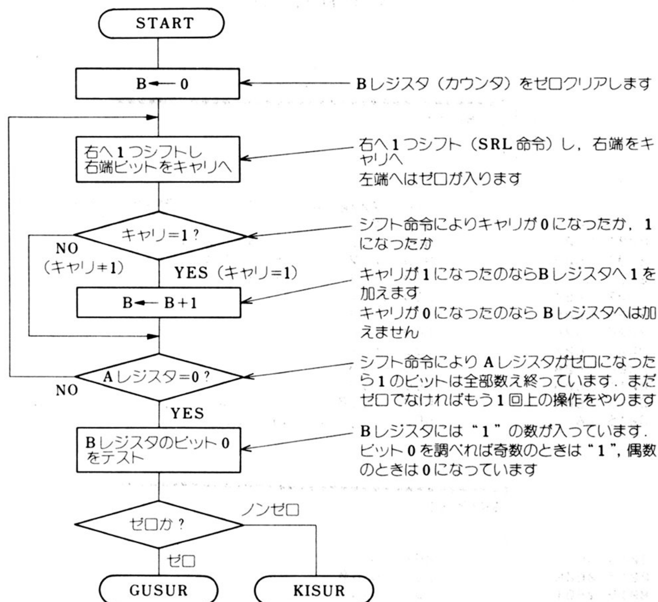
アドレス　マシン語　ラベル　オペコード　オペランド

```
                   ;*********************************
                   ;SAMPLE1
                   ;
                   ;
                          ORG   0000H  ←── このプログラムを0000H番地から配置することをアセンブラに知らせるアセンブラ命令
0000 AF                   XOR   A      ←── Aをゼロにする
0001 C601                 ADD   A,1        A←A+1  Aは1になる
0003 C602                 ADD   A,2        A←A+2  Aは3になる
0005 C603                 ADD   A,3        A←A+3  Aは6になる
0007 C604                 ADD   A,4
0009 C605                 ADD   A,5
000B C606                 ADD   A,6
000D C607                 ADD   A,7
000F C608                 ADD   A,8
0011 C609                 ADD   A,9
0013 C60A                 ADD   A,10   ←── ここで1～10までの和がAレジスタに入っている
0015 76                   HALT         ←── CPUを止める

                   ;*********************************
                   ;SAMPLE2
                   ;
                   ;
0016 AF                   XOR   A
0017 060A                 LD    B,10       カウンタの初期設定
0019 2601                 LD    H,1        ポインタの初期設定
001B 84            LOOP1  ADD   A,H        A←A+H
001C 24                   INC   H          H←H+1
001D 10FC                 DJNZ  LOOP1  ←── B←B−1 {Bがノンゼロなら LOOP1 番地へ，ゼロなら次の番地へ
001F 76                   HALT
```

プログラム 2 （判断）

Aレジスタに入っているデータのビットパターンのうち“1”の数がいくつあるか数えます．結果が偶数ならGUSUR番地へ，奇数ならKISUR番地へジャンプするようにします．〈SAMPLE 3〉



もっとうまい方法があります．AとAのANDをとればAは変わらずに，そのときのAの1の数に応じてパリティ・フラグがセットされます．

```
AND   A            ←── AとAのアンドによりFレジスタをセットします
JP    PE, GUSUR    ←── PE，はパリティ・イーブンすなわちAレジスタの「1の
JP    KISUR            数が偶数であればジャンプせよ」の条件付ジャンプ命令です
```

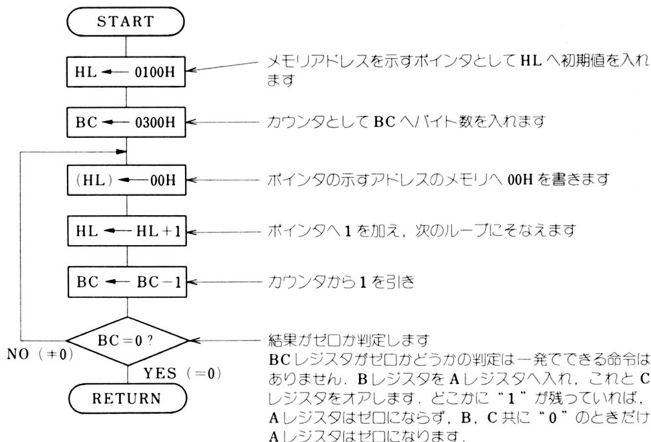
これでも上と同じ働きをします．Fレジスタだけ変わりますが，他のレジスタの内容はこのルーチンに入る前のまま変化しません．

```
                   ;*********************************
                   ;SAMPLE3
                   ;
                   ;
           0200    GUSUR    EQU    0200H
           025F    KISUR    EQU    025FH
     0020  0600             LD     B,0
     0022  CB3F    LOOP2    SRL    A
     0024  3001             JR     NC,JP1
     0026  04               INC    B
     0027  20F9    JP1      JR     NZ,LOOP2
     0029  CB40             BIT    0,B
     002B  CA0002           JP     Z,GUSUR
     002E  C35F02           JP     KISUR
```

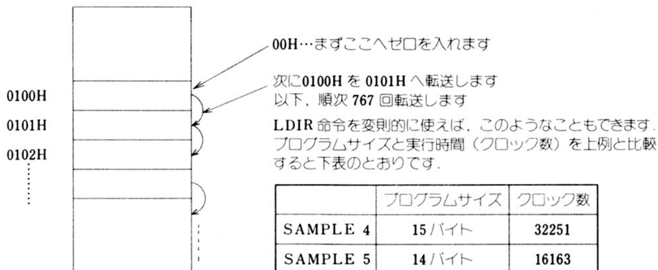
← EQU は GUSUR というラベル名を 0200 H 番地へ結びつけるためのアセンブラ命令

プログラム 3 （メモリクリアサブルーチン）

0100H 番地から 03FFH 番地までの 300H バイト(=768 バイト)の RAM をゼロクリア（すべて 0 で埋めつくす）するサブルーチンを作ります．〈SAMPLE 4〉



別の方法もあります．〈SAMPLE 5〉



LDIR 命令を変則的に使えば，このようなこともできます．プログラムサイズと実行時間（クロック数）を上例と比較すると下表のとおりです．

	プログラムサイズ	クロック数
SAMPLE 4	15 バイト	32251
SAMPLE 5	14 バイト	16163

（プログラムサイズが小さく，実行が早いのがよいプログラムの条件です．ただしあまり名人芸的なプログラムを作るとあとで見たときに何をやっているのかわからなくなることがあります．メンテナンスやデバックをしやすいのもよいプログラムの条件です）

```
                     ;************************************
                     ;SAMPLE4
                     ;
                     ;
0031 210001  MCLEAR  LD    HL,0100H
0034 010003          LD    BC,0300H
0037 3600    ZEROM   LD    (HL),0
0039 23              INC   HL
003A 0B              DEC   BC
003B 78              LD    A,B         } B, C 共にゼロであることを調べる
003C B1              OR    C
003D 20F8            JR    NZ,ZEROM
003F C9              RET
```

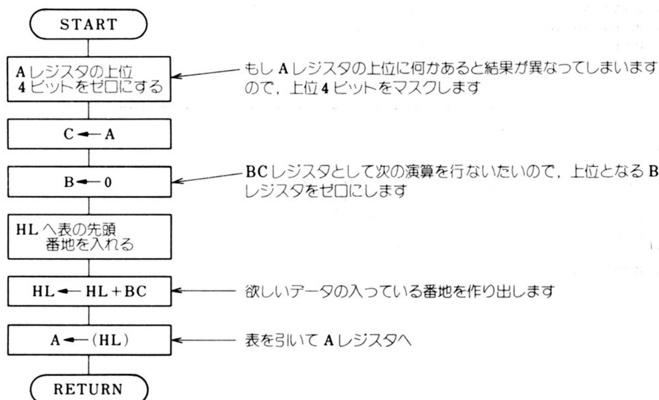
```
                     ;************************************
                     ;SAMPLE5
                     ;
                     ;
0040 210001  MCLR1   LD    HL,0100H
0043 110101          LD    DE,0101H
0046 01FF02          LD    BC,767 ←── { 10進数で定義してよい
                                        マシン語は02FFになっている
0049 3600            LD    (HL),0
004B EDB0            LDIR
004D C9              RET
```

プログラム 4 (変換-テーブルサーチ)

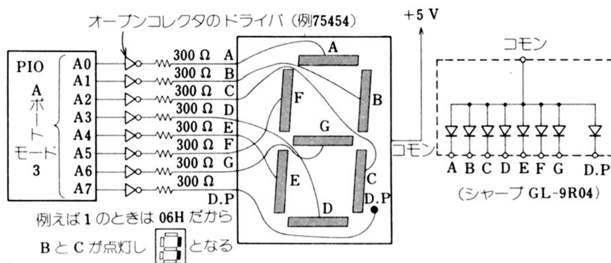
Aレジスタの内容を表に従って変換します(下位4ビット対象). <SAMPLE 6>

Aレジスタ	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
変 換 後	5C	06	5B	4F	66	6D	7D	27	7F	6F	77	7C	39	5E	79	71

変換前のデータが規則的で後は不規則です. したがって, 前をメモリ・アドレスに対応させ, 後をその内容とすれば, 一発で表引きができます. もし相方共不規則なら1つ1つ一致するかどうか見て行くサーチの手法をとらなければなりません.



これは4ビットのデータを数字表示LEDに16進数表示するときに使うサブルーチンです. 多桁表示したいときは, コモンを順次切り替えてダイナミック点灯させます.

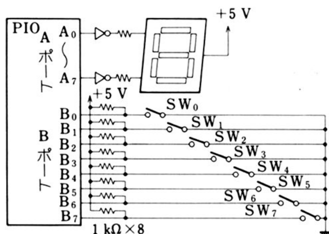


```
                    ;*********** ***************************
                    ;SAMPLE6
                    ;
                    ;
004E E60F    CONV   AND  0FH  ←上位4ビットをマスク
0050 4F             LD   C,A
0051 0600           LD   B,0
0053 215900         LD   HL,TABLE
0056 09             ADD  HL,BC  ←HLに答の入っているアドレスを作成
0057 7E             LD   A,(HL)
0058 C9             RET
0059 5C      TABLE  DEFB 5CH
005A 06             DEFB 06H
005B 5B             DEFB 5BH
005C 4F             DEFB 4FH
005D 66             DEFB 66H
005E 6D             DEFB 6DH
005F 7D             DEFB 7DH
0060 27             DEFB 27H
0061 7F             DEFB 7FH
0062 6F             DEFB 6FH
0063 77             DEFB 77H
0064 7C             DEFB 7CH
0065 39             DEFB 39H
0066 5E             DEFB 5EH
0067 79             DEFB 79H
0068 71             DEFB 71H
```

テーブル（表）
（DEFBはオペランドに書かれた1バイトの値をそのままマシン語としてメモリへ定義するためのアセンブラ命令）

プログラム 5 （スイッチの表示）

PIO に継がれた 8 個のスイッチのパターンを読み，前項のサブルーチンを使って LED に 16 進数表示させます．〈SAMPLE 7〉



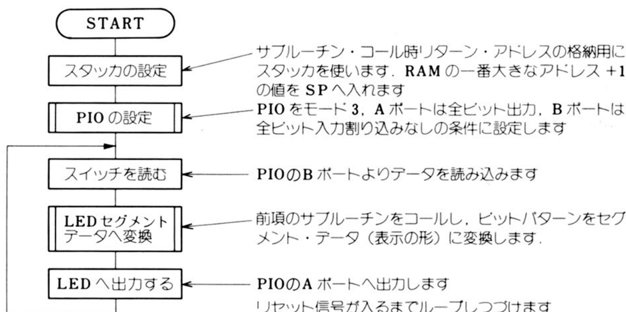
PIO のポートアドレスは下表とします

A ポート　データ	D0
A ポート　コントロール	D1
B ポート　データ	D2
B ポート　コントロール	D3

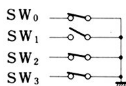
＝“1”

＝“0”

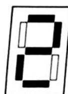
※SW_4～SW_7 は，この例では使わない（無視される）



IN 命令を実行するたびにその時点の SW_0～SW_3 の状態が 16 進表現で LED に表示されます．



なら 0010 だから

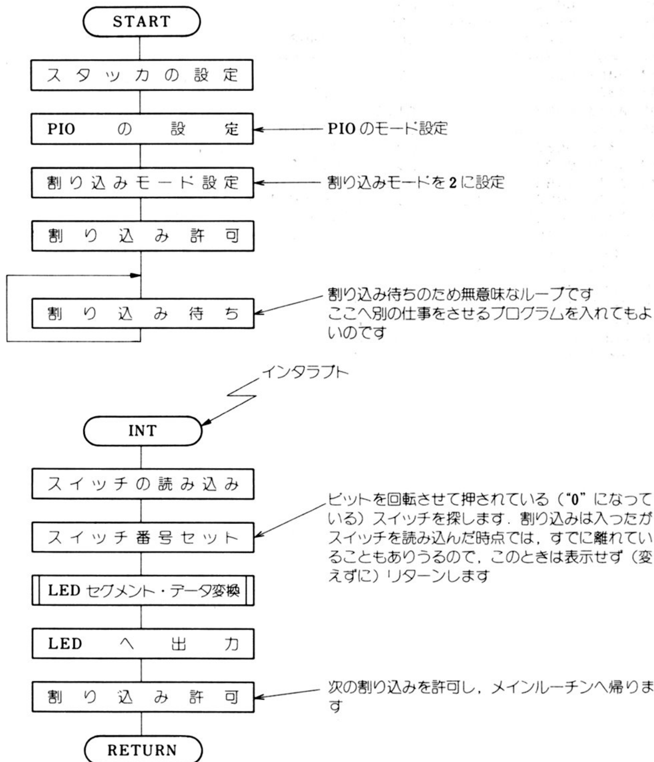


と表示されます

```
                ;****************************************
                ;SAMPLE7
                ;
                ;
          00D0  PIOAD1  EQU   0D0H         ポートアドレスのラベル名定義
          00D1  PIOAC1  EQU   0D1H         EQU は定義のためのアセンブラ命令
          00D2  PIOBD1  EQU   0D2H         (マシン語には変換されない)
          00D3  PIOBC1  EQU   0D3H
0069 310000             LD    SP,0000H  ←── スタッカ設定．FFFF H 番地より前へ
006C 218800             LD    HL,INTPA
006F 0603               LD    B,3          PIO A ポート．制御語の書き込み
0071 0ED1               LD    C,PIOAC1
0073 EDB3               OTIR
0075 218B00             LD    HL,INTPB
0078 0603               LD    B,3          PIO B ポート．制御語の書き込み
007A 0ED3               LD    C,PIOBC1
007C EDB3               OTIR
007E DBD2       LOOP7   IN    A,(PIOBD1)← スイッチ読み込み
0080 CD4E00             CALL  CONV  ←──── データ変換をコール
0083 D3D0               OUT   (PIOAD1),A← 出力
0085 C37E00             JP    LOOP7
0088 CF         INTPA   DEFB  0CFH ←────── モード 3
0089 00                 DEFB  00H  ←────── 全ビット出力
008A 07                 DEFB  07H  ←────── 割り込みなし
008B CF         INTPB   DEFB  0CFH ←────── モード 3
008C FF                 DEFB  0FFH ←────── 全ビット入力
008D 07                 DEFB  07H  ←────── 割り込みなし
```

プログラム 6 （スイッチの表示-割り込み）

前項のスイッチを押ボタンスイッチとして，押されたスイッチ番号（SW_0 なら 0，SW_3 なら 3，SW_7 なら 7）を表示するように割り込みを使ってプログラムを作ります．ただし，同時に押したときは先に押したほうを優先します．全く同時なら番号の小さいほうを優先します．離しても次に押されるまで表示しつづけます．〈SAMPLE 8〉



```
                    ;************ *****************************
                    ;SAMPLE8
                    ;
                    ;
          00D0      PIOAD   EQU   0D0H         ┐
          00D1      PIOAC   EQU   0D1H         │ポートアドレスのラベル名定義
          00D2      PIOBD   EQU   0D2H         │
          00D3      PIOBC   EQU   0D3H         ┘
     008E 310000            LD    SP,0000H      スタッカ設定．FFFF H番地より前へ
     0091 21AC00            LD    HL,INTLPA    ┐
     0094 0603              LD    B,3          │PIO Aポート．制御語の書き込み
     0096 0ED1              LD    C,PIOAC      │
     0098 EDB3              OTIR               ┘
     009A 21AF00            LD    HL,INTLPB    ┐
     009D 0605              LD    B,5          │PIO Bポート．制御語の書き込み
     009F 0ED3              LD    C,PIOBC      │
     00A1 EDB3              OTIR               ┘
     00A3 3E01              LD    A,01H        ┐Iレジスタへ01 H
     00A5 ED47              LD    I,A          ┘
     00A7 ED5E              IM    2 <────────── 割り込みモード2
     00A9 FB                EI  <────────────── 割り込み許可
     00AA 18FE      WAIT    JR    WAIT<──────── 割り込み待ちのループ
     00AC CF        INTLPA  DEFB  0CFH<──────── モード3
     00AD 00                DEFB  00H <──────── 全ビット出力
     00AE 07                DEFB  07H <──────── 割り込みなし
     00AF 00        INTLPB  DEFB  00H <──────── ベクトル00 H
     00B0 CF                DEFB  0CFH <─────── モード3
     00B1 FF                DEFB  0FFH <─────── 全ビット入力
     00B2 97                DEFB  97H <──────── •1 ┐
     00B3 00                DEFB  00H <──────── •2 │［注］下記参照
                            ORG   0100H <────── •3 │
     0100 0201              DEFW  INT <──────── •4 ┘
     0102 DBD2      INT     IN    A,(PIOBD)<─── スイッチの読み込み
     0104 0E00              LD    C,0
     0106 0608              LD    B,8
     0108 1F        LOOP8   RRA <────────────── 右にループし右端ビットをキャリへ
     0109 3006              JR    NC,DISP<───── キャリを調べ"0"のとき表示へ
     010B 0C                INC   C
     010C 10FA              DJNZ  LOOP8
     010E FB                EI                 ┐8ビット共調べて，すべて"1"のときここで
     010F ED4D              RETI               ┘リターン
     0111 79        DISP    LD    A,C
     0112 CD4E00            CALL  CONV <─────── データ変換をコール
     0115 D3D0              OUT   (PIOAD),A<─── LEDへ出力
     0117 FB                EI
     0118 ED4D              RETI
                    ;****** *********************************
                            END <────────────── ソースプログラムの終りをアセンブラに知らせる
                                                アセンブラ命令
```

•1　割り込みあり，マスク制御語あり，オア，ローアクティブ

•2　マスク制御語．全ビット割り込み発生

•3　Iレジスタとベクトルにより示される番地

•4　割り込み処理ルーチンのエントリーアドレス0102番地
　　DEFWはオペランドの2バイトデータを定義するアセンブラ命令

付　録

付1　Z-80　命　令　表

○　n は　　8 ビット定数

○　lm は　16 ビット定数，l が上位 8 ビット，m が下位 8 ビット
（ニーモニックではラベル名を書いてもよい）

○　d は　　−128〜＋127 までのディスプレイスメント

○　e は　　−128〜＋127 までのディスプレイスメント．次の命令の先頭番地を 0 とする．
（ニーモニックではラベル名か絶対番地を書く．ただしアセンブラにより異なる場合もある）．

○　Cy は　　キャリ（C フラグ）

○　添字 $_H$ は上位 8 ビット，添字 $_L$ は下位 8 ビットを意味する．

○　ビット操作命令　SET　RES　BIT　で，ビット番号は下記のとおりである．

7	6	5	4	3	2	1	0

左端 └ 2^7 ビット　　右端 └ 2^0 ビット

○　フラグ変化

× 不定
1 1になる
0 0になる
・ 状態にしたがってセット，リセットされる
− 変化せず

ニーモニック	マシン語	S	Z	H	P	V	N	C	所要クロックサイクル	動作
ADC A, n	CE n								7	8ビット加算キャリ付（アド・ウィズ・キャリ）
ADC A, A	8F									
ADC A, B	88									A←A＋ソース＋Cy
ADC A, C	89									
ADC A, D	8A								4	
ADC A, E	8B	·	·	·	—	·	0	·		
ADC A, H	8C									
ADC A, L	8D									
ADC A, (HL)	8E								7	
ADC A, (IX+d)	DD 8E d								19	
ADC A, (IY+d)	FD 8E d								19	
ADC HL, BC	ED 4A									16ビット加算キャリ付（アド・ウィズ・キャリ）
ADC HL, DE	ED 5A	·	·	×	—	·	0	·	15	
ADC HL, HL	ED 6A									HL←HL＋ソース＋Cy
ADC HL, SP	ED 7A									
ADD A, n	C6 n								7	8ビット加算（アド）
ADD A, A	87									
ADD A, B	80									
ADD A, C	81									A←A＋ソース
ADD A, D	82								4	
ADD A, E	83	·	·	·	—	·	0	·		
ADD A, H	84									
ADD A, L	85									
ADD A, (HL)	86								7	
ADD A, (IX+d)	DD 86 d								19	
ADD A, (IY+d)	FD 86 d								19	
ADD HL, BC	09									16ビット加算（アド）
ADD HL, DE	19								11	
ADD HL, HL	29									HL←HL＋ソース
ADD HL, SP	39									
ADD IX, BC	DD 09									
ADD IX, DE	DD 19									
ADD IX, IX	DD 29	—	—	×	—	—	0	·		IX←IX＋ソース
ADD IX, SP	DD 39									
ADD IY, BC	FD 09								15	
ADD IY, DE	FD 19									
ADD IY, IY	FD 29									IY←IY＋ソース
ADD IY, SP	FD 39									
AND n	E6 n								7	論理積（アンド）
AND A	A7									
AND B	A0									
AND C	A1									A←A∧ソース
AND D	A2								4	
AND E	A3	·	·	1	·	—	0	0		
AND H	A4									
AND L	A5									
AND (HL)	A6								7	
AND (IX+d)	DD A6 d								19	
AND (IY+d)	FD A6 d								19	

（フラグ変化欄の P/V は P と V に分かれる）

a	b	答
0	0	0
0	1	0
1	0	0
1	1	1

ニーモニック	マシン語	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V		N	C		
					P	V				
BIT 0,A	CB 47									ビットテスト
BIT 0,B	CB 40									
BIT 0,C	CB 41									
BIT 0,D	CB 42								8	ソースの第0ビットを調べ
BIT 0,E	CB 43	×	・	1	×	−	0	−		Zフラグを設定
BIT 0,H	CB 44									
BIT 0,L	CB 45									
BIT 0,(HL)	CB 46								12	
BIT 0,(IX+d)	DD CB d 46								20	
BIT 0,(IY+d)	FD CB d 46								20	
BIT 1,A	CB 4F									ビットテスト
BIT 1,B	CB 48									
BIT 1,C	CB 49									ソースの第1ビットを調べ
BIT 1,D	CB 4A								8	Zフラグを設定
BIT 1,E	CB 4B	×	・	1	×	−	0	−		
BIT 1,H	CB 4C									
BIT 1,L	CB 4D									
BIT 1,(HL)	CB 4E								12	
BIT 1,(IX+d)	DD CB d 4E								20	
BIT 1,(IY+d)	FD CB d 4E								20	
BIT 2,A	CB 57									ビットテスト
BIT 2,B	CB 50									
BIT 2,C	CB 51									
BIT 2,D	CB 52								8	ソースの第2ビットを調べ
BIT 2,E	CB 53	×	・	1	×	−	0	−		Zフラグを設定
BIT 2,H	CB 54									
BIT 2,L	CB 55									
BIT 2,(HL)	CB 56								12	
BIT 2,(IX+d)	DD CB d 56								20	
BIT 2,(IY+d)	FD CB d 56								20	
BIT 3,A	CB 5F									ビットテスト
BIT 3,B	CB 58									
BIT 3,C	CB 59									
BIT 3,D	CB 5A								8	ソースの第3ビットを調べ
BIT 3,E	CB 5B	×	・	1	×	−	0	−		Zフラグを設定
BIT 3,H	CB 5C									
BIT 3,L	CB 5D									
BIT 3,(HL)	CB 5E								12	
BIT 3,(IX+d)	DD CB d 5E								20	
BIT 3,(IY+d)	FD CB d 5E								20	
BIT 4,A	CB 67									ビットテスト
BIT 4,B	CB 60									
BIT 4,C	CB 61									
BIT 4,D	CB 62								8	ソースの第4ビットを調べ
BIT 4,E	CB 63	×	・	1	×	−	0	−		Zフラグを設定
BIT 4,H	CB 64									
BIT 4,L	CB 65									
BIT 4,(HL)	CB 66								12	
BIT 4,(IX+d)	DD CB d 66								20	
BIT 4,(IY+d)	FD CB d 66								20	

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	V	N	C	所要クロックサイクル	動作
BIT 5, A	CB 6F	×	·	1	×	−	0	−	8	ビットテスト ソースの第5ビットを調べ Zフラグを設定
BIT 5, B	CB 68									
BIT 5, C	CB 69									
BIT 5, D	CB 6A									
BIT 5, E	CB 6B									
BIT 5, H	CB 6C									
BIT 5, L	CB 6D									
BIT 5, (HL)	CB 6E								12	
BIT 5, (IX+d)	DD CB d 6E								20	
BIT 5, (IY+d)	FD CB d 6E								20	
BIT 6, A	CB 77	×	·	1	×	−	0	−	8	ビットテスト ソースの第6ビットを調べ Zフラグを設定
BIT 6, B	CB 70									
BIT 6, C	CB 71									
BIT 6, D	CB 72									
BIT 6, E	CB 73									
BIT 6, H	CB 74									
BIT 6, L	CB 75									
BIT 6, (HL)	CB 76								12	
BIT 6, (IX+d)	DD CB d 76								20	
BIT 6, (IY+d)	FD CB d 76								20	
BIT 7, A	CB 7F	×	·	1	×	−	0	−	8	ビットテスト ソースの第7ビットを調べ Zフラグを設定
BIT 7, B	CB 78									
BIT 7, C	CB 79									
BIT 7, D	CB 7A									
BIT 7, E	CB 7B									
BIT 7, H	CB 7C									
BIT 7, L	CB 7D									
BIT 7, (HL)	CB 7E								12	
BIT 7, (IX+d)	DD CB d 7E								20	
BIT 7, (IY+d)	FD CB d 7E								20	
CALL NZ, lm	C4 m l	−	−	−	−	−	−	−	成立時 17 不成立 10	サブルーチン・コール(条件付) ・条件が成立すれば戻り番地〔PC〕をスタッカへ PUSH し lm へジャンプ〔PC←lm〕 ・成立しなければ本命令は無視する
CALL Z, lm	CC m l									
CALL NC, lm	D4 m l									
CALL C, lm	DC m l									
CALL PO, lm	E4 m l									
CALL PE, lm	EC m l									
CALL P, lm	F4 m l									
CALL M, lm	FC m l									
CALL lm	CD m l	−	−	−	−	−	−	−	17	サブルーチン・コール(無条件) PC をスタッカへ PUSH し PC← lm
CCF	3F	−	−	×	−	−	0	·	4	Cy を反転〔Cy←$\overline{Cy}$〕
CP n	FE n	·	·	·	−	·	1	·	7	比較（コンペア） A－ソースの演算をする Aの内容は変わらずフラグだけが変化する
CP A	BF								4	
CP B	B8									
CP C	B9									
CP D	BA									
CP E	BB									
CP H	BC									
CP L	BD									
CP (HL)	BE								7	
CP (IX+d)	DD BE d								19	
CP (IY+d)	FD BE d								19	

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	V	N	C	所要クロックサイクル	動作
CPD	ED A9	×	·	×	—	·	1	—	16	比較（コンペア・ディクリメント） A−(HL)のフラグ変化のみ HL←HL−1　BC←BC−1
CPDR	ED B9	×	·	×	—	·	1	—	1バイトにつき 21 最終のみ 16	比較(コンペア・ディクリメント・リピート) CPDをA=(HL)［Zフラグ=1］ または　BC=0　［Vフラグ=0］ までくり返えす
CPI	ED A1	×	·	×	—	·	1	—	16	比較(コンペア・インクリメント) A−(HL)のフラグ変化のみ HL←HL+1　BC←BC−1
CPIR	ED B1	×	·	×	—	·	1	—	1バイトにつき 21 最終のみ 16	比較(コンペア・インクリメント・リピート) CPIをA=(HL)［Zフラグ=1］ または　BC=0　［Vフラグ=0］ までくり返えす
CPL	2F	—	—	1	—	—	1	—	4	コンプリメント Aレジスタに対しビット反転　0→1　1→0
DAA	27	·	·	·	·	—	—	·	4	デジマル・アジャスト・アキュムレータ Aレジスタに対し10進補正
DEC　A	3D	·	·	·	—	·	1	—	4	8ビットデクリメント ソース←ソース−1
DEC　B	05									
DEC　C	0D									
DEC　D	15									
DEC　E	1D									
DEC　H	25									
DEC　L	2D									
DEC　(HL)	35								11	
DEC　(IX+d)	DD 35 d								23	
DEC　(IY+d)	FD 35 d								23	
DEC　BC	0B	—	—	—	—	—	—	—	6	16ビットデクリメント ソース←ソース−1
DEC　DE	1B									
DEC　HL	2B									
DEC　SP	3B									
DEC　IX	DD 2B								10	
DEC　IY	FD 2B								10	
DI	F3	—	—	—	—	—	—	—	4	割り込み禁止(ディセーブル・インタラプト)
DJNZ e	10 e	—	—	—	—	—	—	—	B=0 8 B≠0 13	(デクリメント・ジャンプノンゼロ) B←B−1　B≠0ならeバイトだけジャンプ［PC←PC+e］ B=0ならジャンプせず［e=0と同じ］
EI	FB	—	—	—	—	—	—	—	4	割り込み許可(イネーブル・インタラプト)
EX　(SP), HL	E3	—	—	—	—	—	—	—	19	交換（エクスチェンジ） ソースとディスティネーションの内容を交換する
EX　(SP), IX	DD E3								23	
EX　(SP), IY	FD E3								23	
EX　AF, AF'	08								4	
EX　DE, HL	EB								4	
EXX	D9	—	—	—	—	—	—	—	4	BC　DE　HLの内容と BC' DE' HL' の内容を交換する

ニーモニック	マシン語	S	Z	H	P/V P	P/V V	N	C	所要クロックサイクル	動　　作
HALT	76	—	—	—	—	—	—	—	4	命令実行の進行を止めリセットまたは割り込み待ちとなる（ホルト）
IM　0	ED 46	—	—	—	—	—	—	—	8	割り込みモードを0に設定する
IM　1	ED 56								8	割り込みモードを1に設定する
IM　2	ED 5E								8	割り込みモードを2に設定する
INC　A	3C	·	·	·	—	·	0	—	4	8ビットインクリメント ソース←ソース+1
INC　B	04									
INC　C	0C									
INC　D	14									
INC　E	1C									
INC　H	24									
INC　L	2C									
INC　(HL)	34								11	
INC　(IX+d)	DD 34 d								23	
INC　(IY+d)	FD 34 d								23	
INC　BC	03	—	—	—	—	—	—	—	6	16ビットインクリメント ソース←ソース+1
INC　DE	13									
INC　HL	23									
INC　SP	33									
INC　IX	DD 23								10	
INC　IY	FD 23								10	
IN　A,(C)	ED 78	·	·	0	·	—	0	—	12	入力 Cレジスタの内容番地のポートからディスティネーションのレジスタへ入力 ［ディスティネーション←(C)］
IN　B,(C)	ED 40									
IN　C,(C)	ED 48									
IN　D,(C)	ED 50									
IN　E,(C)	ED 58									
IN　H,(C)	ED 60									
IN　L,(C)	ED 68									
IN　A,(n)	DB n	—	—	—	—	—	—	—	11	入力　n番地のポートからAレジスタへ
IND	ED AA	×	·	×	×	—	×	×	16	イン・ディクリメント (HL)←(C)　HL←HL−1 B←B−1
INDR	ED BA	×	1	×	×	—	×	×	1バイトにつき21 最終のみ16	イン・ディクリメント・リピート INDをB=0までくり返えす
INI	ED A2	×	·	×	×	—	×	×	16	イン・インクリメント (HL)←(C)　HL←HL+1 B←B−1
INIR	ED B2	×	1	×	×	—	×	×	1バイトにつき21 最終のみ16	イン・インクリメント・リピート INIをB=0までくり返えす
JP　(HL)	E9	—	—	—	—	—	—	—	4	ジャンプ（無条件） 各レジスタの内容番地へジャンプ ［PC←HL］
JP　(IX)	DD E9								8	
JP　(IY)	FD E9								8	
JP　lm	C3 m l								10	lm番地へジャンプ［PC←lm］

ニーモニック	マシン語	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V		N	C		
					P	V				
JP NZ, lm	C2 m l									ジャンプ（条件付）
JP Z, lm	CA m l									・条件が成立すれば lm 番地へジャンプ
JP NC, lm	D2 m l									（PC←lm）
JP C, lm	DA m l									・不成立なら本命令は無視する
JP PO, lm	E2 m l	—	—	—	—	—	—	—	10	
JP PE, lm	EA m l									
JP P, lm	F2 m l									
JP M, lm	FA m l									
JR e	18 e									ジャンプ・リラティブ（無条件）
		—	—	—	—	—	—	—	12	eバイト先へジャンプする
										[PC←PC+e]
JR NZ, e	20 e								条件成立	ジャンプ・リラティブ(条件付)
JR Z, e	28 e	—	—	—	—	—	—	—	12	・条件が成立すればeバイト先へジャンプ
JR NC, e	30 e								条件不成立	[PC←PC+e]
JR C, e	38 e								7	・不成立なら本命令は無視する
LD A, n	3E n								7	8ビット転送（ロード）
LD A, A	7F									
LD A, B	78									A←ソース
LD A, C	79									
LD A, D	7A								4	
LD A, E	7B									
LD A, H	7C	—	—	—	—	—	—	—		
LD A, L	7D									
LD A, (lm)	3A m l								13	
LD A, (BC)	0A								7	
LD A, (DE)	1A								7	
LD A, (HL)	7E								7	
LD A, (IX+d)	DD 7E d								19	
LD A, (IY+d)	FD 7E d								19	
LD A, I	ED 57	・	・	0	IFF		0	—	9	8ビット転送　A←I
LD A, R	ED 5F								9	A←R
										IFF：0のとき割り込み禁止（DI） 1のとき割り込み可（EI）になっている LD A, I　LD A, R ではこの値がP/Vにコピーされる
LD B, n	06 n								7	8ビット転送（ロード）
LD B, A	47									
LD B, B	40									B←ソース
LD B, C	41									
LD B, D	42								4	
LD B, E	43	—	—	—	—	—	—	—		
LD B, H	44									
LD B, L	45									
LD B, (HL)	46								7	
LD B, (IX+d)	DD 46 d								19	
LD B, (IY+d)	FD 46 d								19	

ニーモニック	マシン語	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V		N	C		
					P	V				
LD C,n	0E n								7	8ビット転送（ロード）
LD C,A	4F									
LD C,B	48									
LD C,C	49									C←ソース
LD C,D	4A								4	
LD C,E	4B	—	—	—	—	—	—	—		
LD C,H	4C									
LD C,L	4D									
LD C,(HL)	4E								7	
LD C,(IX+d)	DD 4E d								19	
LD C,(IY+d)	FD 4E d								19	
LD D,n	16 n								7	8ビット転送（ロード）
LD D,A	57									
LD D,B	50									
LD D,C	51									D←ソース
LD D,D	52								4	
LD D,E	53	—	—	—	—	—	—	—		
LD D,H	54									
LD D,L	55									
LD D,(HL)	56								7	
LD D,(IX+d)	DD 56 d								19	
LD D,(IY+d)	FD 56 d								19	
LD E,n	1E n								7	8ビット転送（ロード）
LD E,A	5F									
LD E,B	58									
LD E,C	59									E←ソース
LD E,D	5A								4	
LD E,E	5B	—	—	—	—	—	—	—		
LD E,H	5C									
LD E,L	5D									
LD E,(HL)	5E								7	
LD E,(IX+d)	DD 5E d								19	
LD E,(IY+d)	FD 5E d								19	
LD H,n	26 n								7	8ビット転送（ロード）
LD H,A	67									
LD H,B	60									
LD H,C	61									H←ソース
LD H,D	62								4	
LD H,E	63	—	—	—	—	—	—	—		
LD H,H	64									
LD H,L	65									
LD H,(HL)	66								7	
LD H,(IX+d)	DD 66 d								19	
LD H,(IY+d)	FD 66 d								19	

ニーモニック	マシン語	フラグ変化							所要クロックサイクル	動作	
		S	Z	H	P/V		N	C			
					P	V					
LD L, n	2E n								7	8ビット転送（ロード）	
LD L, A	6F								4		
LD L, B	68										
LD L, C	69									L←ソース	
LD L, D	6A										
LD L, E	6B	—	—	—	—	—	—	—			
LD L, H	6C										
LD L, L	6D										
LD L, (HL)	6E								7		
LD L, (IX+d)	DD 6E d								19		
LD L, (IY+d)	FD 6E d								19		
LD I, A	ED 47	—	—	—	—	—	—	—	9	8ビット転送　I←A	
LD R, A	ED 4F								9	R←A	
LD (lm), A	32 m l								13	8ビット転送 (lm)←A	
LD (BC), A	02	—	—	—	—	—	—	—	7	(BC)←A	
LD (DE), A	12								7	(DE)←A	
LD (HL), n	36 n								10	8ビット転送（ロード）	
LD (HL), A	77								7		
LD (HL), B	70										
LD (HL), C	71	—	—	—	—	—	—	—		(HL)←ソース	
LD (HL), D	72										
LD (HL), E	73										
LD (HL), H	74										
LD (HL), L	75										
LD (IX+d), n	DD 36 d n								19	8ビット転送（ロード）	
LD (IX+d), A	DD 77 d										
LD (IX+d), B	DD 70 d										
LD (IX+d), C	DD 71 d	—	—	—	—	—	—	—		(IX+d)←ソース	
LD (IX+d), D	DD 72 d										
LD (IX+d), E	DD 73 d										
LD (IX+d), H	DD 74 d										
LD (IX+d), L	DD 75 d										
LD (IY+d), n	FD 36 d n								19	8ビット転送（ロード）	
LD (IY+d), A	FD 77 d										
LD (IY+d), B	FD 70 d										
LD (IY+d), C	FD 71 d	—	—	—	—	—	—	—		(IY+d)←ソース	
LD (IY+d), D	FD 72 d										
LD (IY+d), E	FD 73 d										
LD (IY+d), H	FD 74 d										
LD (IY+d), L	FD 75 d										
LD BC, lm	01 m l	—	—	—	—	—	—	—	10	16ビット転送	(lm：定数
LD BC, (lm)	ED 4B m l								20	BC←ソース	(lm)：メモリの内容)
LD DE, lm	11 m l	—	—	—	—	—	—	—	10	16ビット転送	メモリからレジスタの場合 たとえば
LD DE, (lm)	ED 5B m l								20	DE←ソース	HL, (lm) では L←(lm)
LD HL, lm	21 m l	—	—	—	—	—	—	—	10	16ビット転送	H←(lm+1)
LD HL, (lm)	2A m l								16	HL←ソース	となる

ニーモニック	マシン語	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V P	P/V V	N	C		
LD　SP, *l* m	31 m *l*	–	–	–	–	–	–	–	10	16ビット転送
LD　SP, (*l* m)	ED 7B m *l*								20	
LD　SP, HL	F9								6	SP←ソース
LD　SP, IX	DD F9								10	
LD　SP, IY	FD F9								10	
LD　IX, *l* m	DD 21 m *l*	–	–	–	–	–	–	–	14	16ビット転送
LD　IX, (*l* m)	DD 2A m *l*								20	IX←ソース
LD　IY, *l* m	FD 21 m *l*	–	–	–	–	–	–	–	14	16ビット転送
LD　IY, (*l* m)	FD 2A m *l*								20	IY←ソース
LD　(*l* m), BC	ED 43 m *l*	–	–	–	–	–	–	–	20	16ビット転送（ロード）
LD　(*l* m), DE	ED 53 m *l*								20	
LD　(*l* m), HL	22 m *l*								16	(*l* m)←ソースL
LD　(*l* m), SP	ED 73 m *l*								20	(*l* m + 1)←ソースH
LD　(*l* m), IX	DD 22 m *l*								20	
LD　(*l* m), IY	FD 22 m *l*								20	
LDD	ED A8	×	×	0	–	·	0	–	16	ブロック転送（ロード・ディクリメント） (DE)←(HL)　DE←DE－1 HL←HL－1　BC←BC－1
LDDR	ED B8	×	×	0	–	0	0	–	1バイトにつき 21 最終のみ 16	ブロック転送 （ロード・ディクリメント・リピート） LDDをBC＝0までくり返えす
LDI	ED A0	×	×	0	–	·	0	–	16	ブロック転送（ロード・インクリメント） (DE)←(HL)　DE←DE＋1 HL←HL＋1　BC←BC－1
LDIR	ED B0	×	×	0	–	0	0	–	1バイトにつき 21 最終のみ 16	ブロック転送 （ロード・インクリメント・リピート） LDIをBC＝0までくり返えす
NEG	ED 44	·	·	·	–	·	1	·	8	ニゲイト　Zの補数をとる A←0－A
NOP	00	–	–	–	–	–	–	–	4	何もしないで次へ （ノーオペレーション）
OR　n	F6 n	·	·	0	·	–	0	0	7	論理和（オア）
OR　A	B7								4	A←A∨ソース
OR　B	B0								4	
OR　C	B1								4	
OR　D	B2								4	
OR　E	B3								4	
OR　H	B4								4	
OR　L	B5								4	
OR　(HL)	B6								7	
OR　(IX+d)	DD B6 d								19	
OR　(IY+d)	FD B6 d								19	

a	b	答
0	0	0
0	1	1
1	0	1
1	1	1

ニーモニック	マシン語	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V		N	C		
					P	V				
OUT (C),A	ED 79									出力
OUT (C),B	ED 41									
OUT (C),C	ED 49									各レジスタの内容をCレジスタの内容番地
OUT (C),D	ED 51	−	−	−	−	−	−	−	12	のポートへ出力
OUT (C),E	ED 59									[(C)←ソース]
OUT (C),H	ED 61									
OUT (C),L	ED 69									
OUT (n),A	D3 n	−	−	−	−	−	−	−	11	出力 Aレジスタの内容をn番地のポートへ
OUTD	ED AB	×	·	×	×	−	×	×	16	アウト・ディクリメント (C)←(HL) HL←HL−1 B←B−1
OTDR	ED BB	×	1	×	×	−	×	×	1バイトにつき 21 最終のみ 16	アウト・ディクリメント・リピート OUTDをB＝0までくり返えす
OUTI	ED A3	×	·	×	×	−	×	×	16	アウト・インクリメント (C)←(HL) HL←HL＋1 B←B−1
OTIR	ED B3	×	1	×	×	−	×	×	1バイトにつき 21 最終のみ 16	アウト・インクリメント・リピート OUTIをB＝0までくり返えす
POP AF	F1									16ビット転送(ポップ)
POP BC	C1									スタッカからレジスタへ転送
POP DE	D1	−	−	−	−	−	−	−	10	ディスティネーション$_L$←(SP)
POP HL	E1									ディスティネーション$_H$←(SP+1)
POP IX	DD E1								14	SP←SP+2
POP IY	FD E1								14	
PUSH AF	F5									16ビット転送(プッシュ)
PUSH BC	C5									レジスタからスタッカへ転送
PUSH DE	D5								11	(SP−1)←ソース$_H$
PUSH HL	E5	−	−	−	−	−	−	−		(SP−2)←ソース$_L$
PUSH IX	DD E5								15	SP←SP−2
PUSH IY	FD E5								15	
RES 0,A	CB 87									ビットリセット
RES 0,B	CB 80									
RES 0,C	CB 81									ソースの第0ビット←0
RES 0,D	CB 82								8	
RES 0,E	CB 83	−	−	−	−	−	−	−		
RES 0,H	CB 84									
RES 0,L	CB 85									
RES 0,(HL)	CB 86								15	
RES 0,(IX+d)	DD CB d 86								23	
RES 0,(IY+d)	FD CB d 86								23	

ニーモニック	マシン語	S	Z	H	P/V	N	C	所要クロックサイクル	動　　作
RES 1, A	CB 8F	–	–	–	–	–	–	8	ビットリセット ソースの第1ビット←0
RES 1, B	CB 88	–	–	–	–	–	–	8	
RES 1, C	CB 89	–	–	–	–	–	–	8	
RES 1, D	CB 8A	–	–	–	–	–	–	8	
RES 1, E	CB 8B	–	–	–	–	–	–	8	
RES 1, H	CB 8C	–	–	–	–	–	–	8	
RES 1, L	CB 8D	–	–	–	–	–	–	8	
RES 1, (HL)	CB 8E	–	–	–	–	–	–	15	
RES 1, (IX+d)	DD CB d 8E	–	–	–	–	–	–	23	
RES 1, (IY+d)	FD CB d 8E	–	–	–	–	–	–	23	
RES 2, A	CB 97	–	–	–	–	–	–	8	ビットリセット ソースの第2ビット←0
RES 2, B	CB 90	–	–	–	–	–	–	8	
RES 2, C	CB 91	–	–	–	–	–	–	8	
RES 2, D	CB 92	–	–	–	–	–	–	8	
RES 2, E	CB 93	–	–	–	–	–	–	8	
RES 2, H	CB 94	–	–	–	–	–	–	8	
RES 2, L	CB 95	–	–	–	–	–	–	8	
RES 2, (HL)	CB 96	–	–	–	–	–	–	15	
RES 2, (IX+d)	DD CB d 96	–	–	–	–	–	–	23	
RES 2, (IY+d)	FD CB d 96	–	–	–	–	–	–	23	
RES 3, A	CB 9F	–	–	–	–	–	–	8	ビットリセット ソースの第3ビット←0
RES 3, B	CB 98	–	–	–	–	–	–	8	
RES 3, C	CB 99	–	–	–	–	–	–	8	
RES 3, D	CB 9A	–	–	–	–	–	–	8	
RES 3, E	CB 9B	–	–	–	–	–	–	8	
RES 3, H	CB 9C	–	–	–	–	–	–	8	
RES 3, L	CB 9D	–	–	–	–	–	–	8	
RES 3, (HL)	CB 9E	–	–	–	–	–	–	15	
RES 3, (IX+d)	DD CB d 9E	–	–	–	–	–	–	23	
RES 3, (IY+d)	FD CB d 9E	–	–	–	–	–	–	23	
RES 4, A	CB A7	–	–	–	–	–	–	8	ビットリセット ソースの第4ビット←0
RES 4, B	CB A0	–	–	–	–	–	–	8	
RES 4, C	CB A1	–	–	–	–	–	–	8	
RES 4, D	CB A2	–	–	–	–	–	–	8	
RES 4, E	CB A3	–	–	–	–	–	–	8	
RES 4, H	CB A4	–	–	–	–	–	–	8	
RES 4, L	CB A5	–	–	–	–	–	–	8	
RES 4, (HL)	CB A6	–	–	–	–	–	–	15	
RES 4, (IX+d)	DD CB d A6	–	–	–	–	–	–	23	
RES 4, (IY+d)	FD CB d A6	–	–	–	–	–	–	23	
RES 5, A	CB AF	–	–	–	–	–	–	8	ビットリセット ソースの第5ビット←0
RES 5, B	CB A8	–	–	–	–	–	–	8	
RES 5, C	CB A9	–	–	–	–	–	–	8	
RES 5, D	CB AA	–	–	–	–	–	–	8	
RES 5, E	CB AB	–	–	–	–	–	–	8	
RES 5, H	CB AC	–	–	–	–	–	–	8	
RES 5, L	CB AD	–	–	–	–	–	–	8	
RES 5, (HL)	CB AE	–	–	–	–	–	–	15	
RES 5, (IX+d)	DD CB d AE	–	–	–	–	–	–	23	
RES 5, (IY+d)	FD CB d AE	–	–	–	–	–	–	23	

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	P/V V	N	C	所要クロックサイクル	動作
RES 6, A	CB B7	—	—	—	—	—	—	—	8	ビットリセット ソースの第6ビット←0
RES 6, B	CB B0									
RES 6, C	CB B1									
RES 6, D	CB B2									
RES 6, E	CB B3									
RES 6, H	CB B4									
RES 6, L	CB B5									
RES 6, (HL)	CB B6								15	
RES 6, (IX+d)	DD CB d B6								23	
RES 6, (IY+d)	FD CB d B6								23	
RES 7, A	CB BF	—	—	—	—	—	—	—	8	ビットリセット ソースの第7ビット←0
RES 7, B	CB B8									
RES 7, C	CB B9									
RES 7, D	CB BA									
RES 7, E	CB BB									
RES 7, H	CB BC									
RES 7, L	CB BD									
RES 7, (HL)	CB BE								15	
RES 7, (IX+d)	DD CB d BE								23	
RES 7, (IY+d)	FD CB d BE								23	
RET	C9	—	—	—	—	—	—	—	10	サブルーチンからのリターン PC へスタックより POP
RET NZ	C0	—	—	—	—	—	—	—	条件成立 11 不成立 5	条件付リターン ・条件が成立すれば POP PC ・不成立なら本命令は無視する
RET Z	C8									
RET NC	D0									
RET C	D8									
RET PO	E0									
RET PE	E8									
RET P	F0									
RET M	F8									
RETI	ED 4D	—	—	—	—	—	—	—	14	割り込み処理からのリターン POP PC
RETN	ED 45	—	—	—	—	—	—	—	14	ノンマスカブル割り込み処理からのリターン POP PC
RLA	17	—	—	0	—	—	0	·	4	ローテート・レフト・アキュムレータ RL A と同じ
RLCA	07	—	—	0	—	—	0	·	4	ローテート・レフト・サーキュラ・アキュムレータ RLC A と同じ
RRA	1F	—	—	0	—	—	0	·	4	ローテート・ライト・アキュムレータ RR A と同じ
RRCA	0F	—	—	0	—	—	0	·	4	ローテート・ライト・サーキュラ・アキュムレータ RRC A と同じ

ニーモニック	マシン語	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V: P	P/V: V	N	C		
RL A	CB 17									ローテート・レフト
RL B	CB 10									
RL C	CB 11									
RL D	CB 12								8	
RL E	CB 13	·	·	0	·	—	0	·		Cy ← 7 0
RL H	CB 14									
RL L	CB 15									
RL (HL)	CB 16								15	
RL (IX+d)	DD CB d 16								23	
RL (IY+d)	FF CB d 16								23	
RLC A	CB 07									ローテート・レフト・サーキュラ
RLC B	CB 00									
RLC C	CB 01									
RLC D	CB 02								8	
RLC E	CB 03	·	·	0	·	—	0	·		Cy ← 7 0
RLC H	CB 04									
RLC L	CB 05									
RLC (HL)	CB 06								15	
RLC (IX+d)	DD CB d 06								23	
RLC (IY+d)	FD CB d 06								23	
RR A	CB 1F									ローテート・ライト
RR B	CB 18									
RR C	CB 19									
RR D	CB 1A								8	
RR E	CB 1B	·	·	0	·	—	0	·		→ Cy → 7 0
RR H	CB 1C									
RR L	CB 1D									
RR (HL)	CB 1E								15	
RR (IX+d)	DD CB d 1E								23	
RR (IY+d)	FD CB d 1E								23	
RRC A	CB 0F									ローテート・ライト・サーキュラ
RRC B	CB 08									
RRC C	CB 09									
RRC D	CB 0A								8	
RRC E	CB 0B	·	·	0	·	—	0	·		→ Cy → 7 0
RRC H	CB 0C									
RRC L	CB 0D									
RRC (HL)	CB 0E								15	
RRC (IX+d)	DD CB d 0E								23	
RRC (IY+d)	FD CB d 0E								23	
RLD	ED 6F	·	·	0	·	—	0	—	18	ローテート・レフト・ディジット A (HL) 7 4 3 0 7 4 3 0
RRD	ED 67	·	·	0	·	—	0	—	18	ローテート・ライト・ディジット A (HL) 7 4 3 0 7 4 3 0

ニーモニック	マシン語	S	Z	H	P	V	N	C	所要クロックサイクル	動作
		フラグ変化			P/V					
RST 00H	C7	—	—	—	—	—	—	—	11	リスタート
RST 08H	CF									
RST 10H	D7									0000H~0038Hのいずれかの番地
RST 18H	DF									に対するCALL
RST 20H	E7									
RST 28H	EF									
RST 30H	F7									
RST 38H	FF									
SBC A, n	DE n	·	·	·	—	·	1	·	7	8ビット引き算（キャリ付）
SBC A, A	9F								4	（サブトラクト・ウィズ・キャリ）
SBC A, B	98									
SBC A, C	99									A←A－ソース－Cy
SBC A, D	9A									
SBC A, E	9B									
SBC A, H	9C									
SBC A, L	9D									
SBC A, (HL)	9E								7	
SBC A, (IX+d)	DD 9E d								19	
SBC A, (IY+d)	FD 9E d								19	
SBC HL, BC	ED 42	·	·	×	—	·	1	·	15	16ビット引き算（キャリ付）
SBC HL, DE	ED 52									（サブトラクト・ウィズ・キャリ）
SBC HL, HL	ED 62									HL←HL－ソース－Cy
SBC HL, SP	ED 72									
SCF	37	—	—	0	—	—	0	1	4	セット・キャリフラグ　Cy←1
SET 0, A	CB C7	—	—	—	—	—	—	—	8	ビットセット
SET 0, B	CB C0									
SET 0, C	CB C1									
SET 0, D	CB C2									ソースの第0ビット←1
SET 0, E	CB C3									
SET 0, H	CB C4									
SET 0, L	CB C5									
SET 0, (HL)	CB C6								15	
SET 0, (IX+d)	DD CB d C6								23	
SET 0, (IY+d)	FD CB d C6								23	
SET 1, A	CB CF	—	—	—	—	—	—	—	8	ビットセット
SET 1, B	CB C8									
SET 1, C	CB C9									
SET 1, D	CB CA									ソースの第1ビット←1
SET 1, E	CB CB									
SET 1, H	CB CC									
SET 1, L	CB CD									
SET 1, (HL)	CB CE								15	
SET 1, (IX+d)	DD CB d CE								23	
SET 1, (IY+d)	FD CB d CE								23	

ニーモニック	マシン語	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V		N	C		
					P	V				
SET 2,A	CB D7									ビットセット
SET 2,B	CB D0									
SET 2,C	CB D1									
SET 2,D	CB D2								8	ソースの第2ビット←1
SET 2,E	CB D3	—	—	—	—	—	—	—		
SET 2,H	CB D4									
SET 2,L	CB D5									
SET 2,(HL)	CB D6								15	
SET 2,(IX+d)	DD CB d D6								23	
SET 2,(IY+d)	FD CB d D6								23	
SET 3,A	CB DF									ビットセット
SET 3,B	CB D8									
SET 3,C	CB D9									
SET 3,D	CB DA								8	ソースの第3ビット←1
SET 3,E	CB DB	—	—	—	—	—	—	—		
SET 3,H	CB DC									
SET 3,L	CB DD									
SET 3,(HL)	CB DE								15	
SET 3,(IX+d)	DD CB d DE								23	
SET 3,(IY+d)	FD CB d DE								23	
SET 4,A	CB E7									ビットセット
SET 4,B	CB E0									
SET 4,C	CB E1									
SET 4,D	CB E2								8	ソースの第4ビット←1
SET 4,E	CB E3	—	—	—	—	—	—	—		
SET 4,H	CB E4									
SET 4,L	CB E5									
SET 4,(HL)	CB E6								15	
SET 4,(IX+d)	DD CB d E6								23	
SET 4,(IY+d)	FD CB d E6								23	
SET 5,A	CB EF									ビットセット
SET 5,B	CB E8									
SET 5,C	CB E9									
SET 5,D	CB EA								8	ソースの第5ビット←1
SET 5,E	CB EB	—	—	—	—	—	—	—		
SET 5,H	CB EC									
SET 5,L	CB ED									
SET 5,(HL)	CB EE								15	
SET 5,(IX+d)	DD CB d EE								23	
SET 5,(IY+d)	FD CB d EE								23	
SET 6,A	CB F7									ビットセット
SET 6,B	CB F0									
SET 6,C	CB F1									
SET 6,D	CB F2								8	ソースの第6ビット←1
SET 6,E	CB F3	—	—	—	—	—	—	—		
SET 6,H	CB F4									
SET 6,L	CB F5									
SET 6,(HL)	CB F6								15	
SET 6,(IX+d)	DD CB d F6								23	
SET 6,(IY+d)	FD CB d F6								23	

ニーモニック	マシン語	S	Z	H	P	V	N	C	所要クロックサイクル	動　作
		フラグ変化			P/V					
SET 7, A	CB FF	—	—	—	—	—	—	—	8	ビットセット ソースの第7ビット←1
SET 7, B	CB F8									
SET 7, C	CB F9									
SET 7, D	CB FA									
SET 7, E	CB FB									
SET 7, H	CB FC									
SET 7, L	CB FD									
SET 7, (HL)	CB FE								15	
SET 7, (IX+d)	DD CB d FE								23	
SET 7, (IY+d)	FD CB d FE								23	
SLA A	CB 27	·	·	0	·	—	0	·	8	シフト・レフト・アリスメチック Cy ← 7　0 ← 0
SLA B	CB 20									
SLA C	CB 21									
SLA D	CB 22									
SLA E	CB 23									
SLA H	CB 24									
SLA L	CB 25									
SLA (HL)	CB 26								15	
SLA (IX+d)	DD CB d 26								23	
SLA (IY+d)	FD CB d 26								23	
SRA A	CB 2F	·	·	0	·	—	0	·	8	シフト・ライト・アリスメチック Cy　7　0
SRA B	CB 28									
SRA C	CB 29									
SRA D	CB 2A									
SRA E	CB 2B									
SRA H	CB 2C									
SRA L	CB 2D									
SRA (HL)	CB 2E								15	
SRA (IX+d)	DD CB d 2E								23	
SRA (IY+d)	FD CB d 2E								23	
SRL A	CB 3F	·	·	0	·	—	0	·	8	シフト・ライト・ロジカル Cy　7　0 0
SRL B	CB 38									
SRL C	CB 39									
SRL D	CB 3A									
SRL E	CB 3B									
SRL H	CB 3C									
SRL L	CB 3D									
SRL (HL)	CB 3E								15	
SRL (IX+d)	DD CB d 3E								23	
SRL (IY+d)	FD CB d 3E								23	
SUB n	D6 n	·	·	·	—	·	1	·	7	8ビット引き算 （サブトラクト） A←A－ソース
SUB A	97								4	
SUB B	90									
SUB C	91									
SUB D	92									
SUB E	93									
SUB H	94									
SUB L	95									
SUB (HL)	96								7	
SUB (IX+d)	DD 96 d								19	
SUB (IY+d)	FD 96 d								19	

ニーモニック	マシン語	S	Z	H	P	V	N	C	所要クロックサイクル	動作
XOR n	EE n								7	排他的論理和(エクスクルーシブ・オア)
XOR A	AF								4	
XOR B	A8								4	
XOR C	A9								4	A←A⊕ソース
XOR D	AA								4	
XOR E	AB	·	·	O	·	—	O	O	4	
XOR H	AC								4	
XOR L	AD								4	
XOR (HL)	AE								7	
XOR (IX+d)	DD AE d								19	
XOR (IY+d)	FD AE d								19	

a	b	答
0	0	0
0	1	1
1	0	1
1	1	0

フラグ変化表

	命令		フラグ							コメント
			S	Z	H	P/V		N	C	
						P	V			
1	8ビット加算	ADD ADC	·	·	·	—	·	0	·	
2	8ビット減算系	SUB SBC CP NEG	·	·	·	—	·	1	·	
3	論理積	AND	·	·	1	·	—	0	0	Cフラグを0にするとき
4	論理和	OR XOR	·	·	0	·	—	0	0	
5	8ビットインクリメント	INC	·	·	·	—	·	0	—	16ビットINC DECでは変化しない
6	8ビットデクリメント	DEC	·	·	·	—	·	1	—	
7	16ビット加算	ADD	—	—	×	—	—	0	·	
8	16ビットキャリ付加算	ADC	·	·	×	—	·	0	·	
9	16ビットキャリ付減算	SBC	·	·	×	—	·	1	·	
10	ローテート・アキュムレータ	RLA RLCA RRA RRCA	—	—	0	—	—	0	·	
11	ローテート	RL RLC RR RRC	·	·	0	·	—	0	·	
12	シフト	SLA SRA SRL	·	·	0	·	—	0	·	
13	ローテート・ディジット	RLD RRD	·	·	0	·	—	0	—	
14	10進補正	DAA	·	·	·	·	—	—	·	
15	ビット反転	CPL	—	—	1	—	—	1	—	
16	セットキャリ	SCF	—	—	0	—	—	0	1	
17	キャリ反転	CCF	—	—	×	—	—	0	·	
18	間接アドレシング入力	IN r, (C)	·	·	0	·	—	0	—	IN A, (n)では変化しない
19	ブロック入力出力	INI IND OUTI OUTD	×	·	×	×	—	×	×	
20	リピート入力出力	INIR INDR OTIR OTDR	×	1	×	×	—	×	×	
21	ブロック転送	LDI LDD	×	×	0	—	·	0	—	
22	リピート転送	LDIR LDDR	×	×	0	—	0	0	—	
23	ブロックサーチ	CPI CPD CPIR CPDR	×	·	×	—	·	1	—	
24	Iレジスタ, Rレジスタ	LD A, I LD A, R	·	·	0	IFF		0	—	P/VにIFFがコピーされる
25	ビットテスト	BIT	×	·	1	×	—	0	—	

×　不定
1　1になる
0　0になる
·　状態にしたがってセット, リセットされる
—　変化せず

IFF：0のとき割り込み禁止(DI)
1のとき割り込み可(EI)
になっている
LD A, I　LD A, R
ではこの値がP/Vにコピーされる

付2　Z-80　規　格　表（参考）

(1)　CPU

絶対最大定格

項　　　目	記　号	定　格　値	単　位
入　力　電　圧	V_{IN}	−0.3 ～ +7	V
出　力　電　圧	V_{OUT}	−0.3 ～ +7	V
動　作　温　度	T_{opr}	0 ～ +70	℃
保　存　温　度	T_{stg}	−65 ～ +150	℃

電気的特性

DC 特性

(Ta ＝ 0 ℃～＋70 ℃，V_{CC} ＝＋5 V ± 5 %)

記　号	項　　　目	最小値	最大値	単位	測定条件
V_{ILC}	クロック“L”入力電圧	−0.3	0.45	V	
V_{IHC}	クロック“H”入力電圧	$V_{CC}-0.6$	$V_{CC}+0.3$	V	
V_{IL}	“L”入力電圧	−0.3	0.8	V	
V_{IH}	“H”入力電圧	2.0	V_{CC}	V	
V_{OL}	“L”出力電圧		0.4	V	I_{OL} ＝ 1.8mA
V_{OH}	“H”出力電圧	2.4		V	I_{OH} ＝ −250μA
I_{CC}	消費電流		150 200	mA	(上段　Z-80 CPU 下段　Z-80A CPU)
I_{LI}	入力リーク電流		10	μA	$V_{IN}=0\sim V_{CC}$
I_{LOH}	トライステート出力リーク電流		10	μA	$V_{OUT}=2.4V\sim V_{CC}$
I_{LOL}	トライステート出力リーク電流		−10	μA	V_{OUT} ＝ 0.4 V
I_{LD}	入力時のデータ・バスのリーク電流		±10	μA	$0\leqq V_{IN}\leqq V_{CC}$

端子容量

(Ta ＝＋25 ℃，f ＝ 1 MHz)

記　号	項　　　目	最大値	単位	測定条件
C_{Φ}	クロック入力容量	50	pF	被測定端子以外のすべての端子は接地
C_{IN}	入力容量	8	pF	
C_{OUT}	出力容量	12	pF	

AC 特性

(Ta ＝0℃～＋70℃，V_{CC} ＝＋5V±5%)

信号	記号	パラメータ	Z-80 CPU 最小値	Z-80 CPU 最大値	Z-80A CPU 最小値	Z-80A CPU 最大値	単位	測定条件
Φ	t_C	クロック周期	0.4	200	0.25	200	μs	
	$t_{W(\Phi H)}$	クロック・パルス幅("H")	180		110		ns	
	$t_{W(\Phi L)}$	クロック・パルス幅("L")	180	2000	110	2000	ns	
	t_r, t_f	クロックの立ち上がり・立ち下がり時間		30		30	ns	
A_0-A_{15}	$t_D(AD)$	クロックの立ち上がりから出力までの遅延		145		110	ns	C_L=50pF
	$t_F(AD)$	出力がフロート状態になるまでの遅延		110		90	ns	
	t_{acm}	$\overline{MREQ}$に先立つ出力確定時間 (メモリ・サイクル)	〔1〕		〔1〕		ns	
	t_{aci}	$\overline{IORQ}$，$\overline{RD}$または$\overline{WR}$に先立つ出力確定時間 (入出力サイクル)	〔2〕		〔2〕		ns	
	t_{ca}	$\overline{RD}$，$\overline{WR}$，$\overline{IORQ}$または$\overline{MREQ}$からの出力保持時間	〔3〕		〔3〕		ns	
	t_{caf}	$\overline{RD}$または$\overline{WR}$からの出力保持時間 (フロート状態への遷移時)	〔4〕		〔4〕		ns	
D_0-D_7	$t_D(D)$	クロックの立ち下がりから出力までの遅延		230		150	ns	C_L=50pF
	$t_F(D)$	出力がフロート状態になるまでの遅延 (書き込みサイクル)		90		90	ns	
	$t_{S\Phi}(D)$	クロックの立ち上がりに対するセットアップ時間 (M1サイクル)	50		35		ns	
	$t_{S\overline{\Phi}}(D)$	クロックの立ち下がりに対するセットアップ時間 (M2～M5サイクル)	60		50		ns	
	t_{dcm}	$\overline{WR}$に先立つ出力確定時間 (メモリ・サイクル)	〔5〕		〔5〕		ns	
	t_{dci}	$\overline{WR}$に先立つ出力確定時間 (入出力サイクル)	〔6〕		〔6〕		ns	
	t_{cdf}	$\overline{WR}$からの出力保存時間	〔7〕		〔7〕		ns	
	t_H	ホールド時間	0		0		ns	
$\overline{MREQ}$	$t_{DL\overline{\Phi}}(MR)$	クロックの立ち下がりから$\overline{MREQ}$＝"L"になるまでの遅延		100		85	ns	C_L=50pF
	$t_{DH\Phi}(MR)$	クロックの立ち上がりから$\overline{MREQ}$＝"H"になるまでの遅延 (M1サイクル)		100		85	ns	

信　号	記　号	パ　ラ　メ　ー　タ	Z-80 CPU		Z-80A CPU		単位	測定条件
			最小値	最大値	最小値	最大値		
$\overline{MREQ}$	$t_{DH\bar{\Phi}}$(MR)	クロックの立ち下がりから$\overline{MREQ}$＝“H”になるまでの遅延（M２～M５サイクル）		100		85	ns	C_L=50pF
	$t_W(\overline{MR_L})$	$\overline{MREQ}$のパルス幅(“L”)	〔8〕		〔8〕		ns	
	$t_W(\overline{MR_H})$	$\overline{MREQ}$のパルス幅(“H”)	〔9〕		〔9〕		ns	
$\overline{IORQ}$	$t_{DL\Phi}$(IR)	クロックの立ち上がりから$\overline{IORQ}$＝“L”になるまでの遅延（入出力サイクル）		90		75	ns	C_L=50pF
	$t_{DL\bar{\Phi}}$(IR)	クロックの立ち下がりから$\overline{IORQ}$＝“L”になるまでの遅延（INTAサイクル）		110		85	ns	
	$t_{DH\Phi}$(IR)	クロックの立ち上がりから$\overline{IORQ}$＝“H”になるまでの遅延（INTAサイクル）		100		85	ns	
	$t_{DH\bar{\Phi}}$(IR)	クロックの立ち下がりから$\overline{IORQ}$＝“H”になるまでの遅延（入出力サイクル）		110		85	ns	
$\overline{RD}$	$t_{DL\Phi}$(RD)	クロックの立ち上がりから$\overline{RD}$＝“L”になるまでの遅延（入出力サイクル）		100		85	ns	C_L=50pF
	$t_{DL\bar{\Phi}}$(RD)	クロックの立ち下がりから$\overline{RD}$＝“L”になるまでの遅延（メモリ・サイクル）		130		95	ns	
	$t_{DH\Phi}$(RD)	クロックの立ち上がりから$\overline{RD}$＝“H”になるまでの遅延（M１サイクル）		100		85	ns	
	$t_{DH\bar{\Phi}}$(RD)	クロックの立ち下がりから$\overline{RD}$＝“H”になるまでの遅延（M２～M５サイクル）		110		85	ns	
$\overline{WR}$	$t_{DL\Phi}$(WR)	クロックの立ち上がりから$\overline{WR}$＝“L”になるまでの遅延（入出力サイクル）		80		65	ns	C_L=50pF
	$t_{DL\bar{\Phi}}$(WR)	クロックの立ち下がりから$\overline{WR}$＝“L”になるまでの遅延（メモリ・サイクル）		90		80	ns	
	$t_{DH\bar{\Phi}}$(WR)	クロックの立ち下がりから$\overline{WR}$＝“H”になるまでの遅延		100		80	ns	
	$t_W(\overline{WR_L})$	$\overline{WR}$のパルス幅(“L”)	〔10〕		〔10〕		ns	
$\overline{M1}$	t_{DL}(M1)	クロックの立ち上がりから$\overline{M1}$＝“L”になるまでの遅延		130		100	ns	C_L=50pF
	t_{DH}(M1)	クロックの立ち上がりから$\overline{M1}$＝“H”になるまでの遅延		130		100	ns	
$\overline{RFSH}$	t_{DL}(RF)	クロックの立ち上がりから$\overline{RFSH}$＝“L”になるまでの遅延		180		130	ns	C_L=50pF

信号	記号	パラメータ	Z-80 CPU 最小値	Z-80 CPU 最大値	Z-80A CPU 最小値	Z-80A CPU 最大値	単位	測定条件
$\overline{RFSH}$	$t_{DH}(RF)$	クロックの立ち上がりから$\overline{RFSH}$＝“H”になるまでの遅延		150		120	ns	C_L＝50pF
$\overline{WAIT}$	t_S (WT)	クロックの立ち下がりに対するセットアップ時間	70		70		ns	
$\overline{HALT}$	t_D (HT)	クロックの立ち下がりからの遅延		300		300	ns	C_L＝50pF
$\overline{INT}$	t_S (IT)	クロックの立ち上がりに対するセットアップ時間	80		80		ns	
$\overline{NMI}$	$t_W(\overline{NM_L})$	$\overline{NMI}$のパルス幅(“L”)	80		80		ns	
$\overline{BUSRQ}$	t_S (BQ)	クロックの立ち上がりに対するセットアップ時間	80		50		ns	
$\overline{BUSAK}$	t_{DL}(BA)	クロックの立ち上がりから$\overline{BUSAK}$＝“L”になるまでの遅延		120		100	ns	C_L＝50pF
	t_{DH}(BA)	クロックの立ち下がりから$\overline{BUSAK}$＝“H”になるまでの遅延		110		100	ns	
$\overline{RESET}$	t_S (RS)	クロックの立ち上がりに対するセットアップ時間	90		60		ns	
	t_F (C)	フロート状態になるまでの遅延 ($\overline{MREQ}$，$\overline{IORQ}$，$\overline{RD}$および$\overline{WR}$)		100		80	ns	
	t_{mr}	$\overline{IORQ}$に先立つ$\overline{M1}$出力(“L”)の確定時間 (INTAサイクル)	〔11〕		〔11〕		ns	

注〔1〕 $t_{acm}=t_W(\Phi_H)+tf-75$
〔2〕 $t_{aci}=t_C-80$
〔3〕 $t_{ca}=t_W(\Phi_L)+t_r-40$
〔4〕 $t_{caf}=t_W(\Phi_L)+t_r-60$
〔5〕 $t_{dcm}=t_C-210$
〔6〕 $t_{dci}=t_W(\Phi_L)+t_r-210$
〔7〕 $t_{cdf}=t_W(\Phi_L)+t_r-80$
〔8〕 $t_W(\overline{MR}_L)=t_C-40$
〔9〕 $t_W(\overline{MR}_H)=t_W(\Phi_H)+tf-30$
〔10〕 $t_W(\overline{WR}_L)=t_C-40$
〔11〕 $t_{mr}=2t_C+t_W(\Phi_H)tf-80$

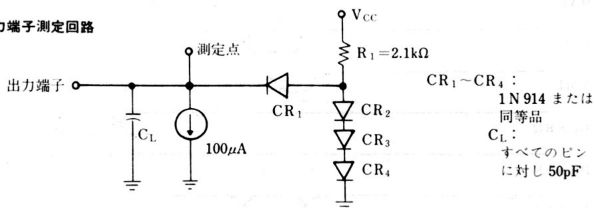
○データを$\overline{RD}$に同期してバスに送り出すことが望ましい。割り込みアクノリッジ・サイクルでは$\overline{M1}$および$\overline{IORQ}$の両方に同期して送り出すことが望ましい。

○制御信号はすべて内部で同期がとれているため、クロックについて非同期的に使用してもよい。

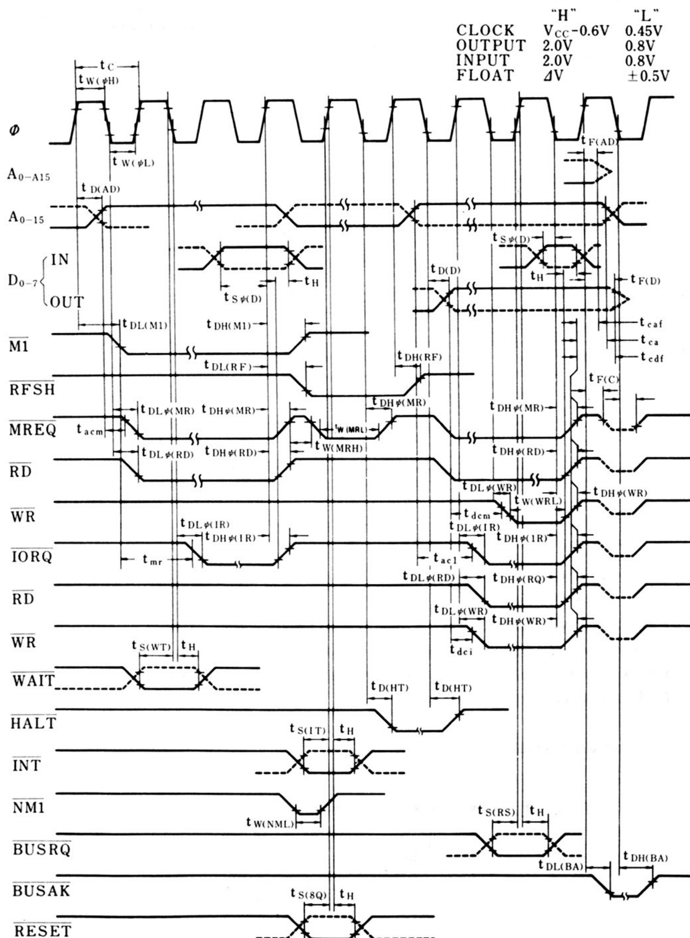
○Ta＝＋70℃、V_{CC}＝＋5V±5％における負荷容量と出力の遅延との関係は次のとおりです。
負荷容量の50pF増加につき遅延は10ns増加します。負荷容量の最大値は、データ・バスが200pFで、他は100pFです。

○$\overline{RESET}$の入力幅は最低3クロック・サイクル必要です。

出力端子測定回路



AC タイミング図


"H"
"L"
CLOCK
V_{CC}-0.6V
0.45V
OUTPUT
2.0V
0.8V
INPUT
2.0V
0.8V
FLOAT
ΔV
±0.5V
Φ
A_{0-A15}
A_{0-15}
D_{0-7}
IN
OUT
M1
RFSH
MREQ
RD
WR
IORQ
RD
WR
WAIT
HALT
INT
NM1
BUSRQ
BUSAK
RESET

（2） PIO

絶対最大定格

項　　　目	記　号	定　格　値	単　位
入力電圧	V_{IN}	−0.3 ～ +7	V
出力電圧	V_{OUT}	−0.3 ～ +7	V
動作温度	T_{opr}	0 ～ +70	℃
保存温度	T_{stg}	−65 ～ +150	℃

電気的特性

DC 特性　　(Ta ＝0℃～＋ 70 ℃，V_{CC}＝＋5 V ± 5％)

記　号	項　　　目	最小値	最大値	単位	測定条件
V_{ILC}	クロック“L”入力電圧	−0.3	0.45	V	
V_{IHC}	クロック“H”入力電圧	V_{CC} −0.6	V_{CC} +0.3	V	
V_{IL}	“L”入力電圧	−0.3	0.8	V	
V_{IH}	“H”入力電圧	2.0	V_{CC}	V	
V_{OL}	“L”出力電圧		0.4	V	I_{OL}＝ 2mA
V_{OH}	“H”出力電圧	2.4		V	I_{OH}＝−250μA
I_{CC}	消費電流		70	mA	
I_{LI}	入力リーク電流		10	μA	V_{IN}＝0～V_{CC}
I_{LOH}	トライステート出力リーク電流		10	μA	V_{OUT}＝2.4V～V_{CC}
I_{LOL}	トライステート出力リーク電流		−10	μA	V_{OUT}＝0.4 V
I_{LD}	入力時のデータ・バスのリーク電流		±10	μA	$0 \leqq V_{IN} \leqq V_{CC}$
I_{OHD}	ダーリントン駆動電流	−1.5		mA	V_{OH}＝1.5V ポートBのみ

端子容量

(Ta ＝＋25 ℃，f ＝1 MHz)

記　号	項　　　目	最大値	単　位	測　定　条　件
C_{Φ}	クロック入力容量	12	pF	被測定端子以外のすべての端子は接地
C_{IN}	入力容量	7	pF	
C_{OUT}	出力容量	10	pF	

AC 特性

（Ta＝0℃～＋70℃、V_{CC}＝＋5V±5%）

信号	記号	パラメータ	Z-80 PIO 最小値	Z-80 PIO 最大値	Z-80A PIO 最小値	Z-80A PIO 最大値	単位	測定条件
Φ	t_C	クロック周期	400	〔1〕	250	〔1〕	ns	
	t_W (ΦH)	クロック・パルス幅（"H"）	170	2000	105	2000	ns	
	t_W (ΦL)	クロック・パルス幅（"L"）	170	2000	105	2000	ns	
	t_r, t_f	クロック立ち上がり・立ち下がり時間		30		30	ns	
	t_H	ホールド時間	0		0		ns	
$\overline{CE}$, C/D, B/A	$t_{S\Phi}$ (CS)	読み出しまたは書き込みサイクルの制御信号のセットアップ時間	280		145		ns	
D_0-D_7	t_{DR} (D)	$\overline{RD}$の立ち下がりからデータ出力までの遅延		430〔2〕		380〔2〕	ns	C_L＝50pF
	$t_{S\Phi}$ (D)	書き込みまたはM1サイクルのデータのセットアップ時間	50		50		ns	
	t_{DI} (D)	INTAサイクルの$\overline{IORQ}$の立ち下がりからデータ出力までの遅延		340〔2〕		250〔2〕	ns	
	t_F (D)	$\overline{RD}$または$\overline{IORQ}$の立ち上がりから出力バッファ・フロートまでの遅延		160		110	ns	
IEI	t_S (IEI)	INTAサイクルの$\overline{IORQ}$の立ち下がりに対するセットアップ時間	140		140		ns	
IEO	t_{DH} (IO)	IEIの立ち上がりからの遅延		210〔4〕		160〔4〕	ns	C_L＝50pF
	t_{DL} (IO)	IEIの立ち下がりからの遅延（注1）		190〔4〕		130〔4〕	ns	
	t_{DM} (IO)	$\overline{M1}$の立ち下がりからの遅延（M1サイクルの直前で割り込みが発生したとき）		300〔4〕		190〔4〕	ns	
$\overline{IORQ}$	$t_{S\Phi}$ (IR)	読み出しまたは書き込みサイクルのセットアップ時間	250		115		ns	
$\overline{M1}$	$t_{S\Phi}$ (M1)	INTAまたはM1サイクルのセットアップ時間	210		90		ns	
$\overline{RD}$	$t_{S\Phi}$ (RD)	読み出しまたはM1サイクルのセットアップ時間	240		115		ns	
$\overline{INT}$	t_D (IT)	$\overline{STB}$の立ち上がりからの遅延		490		440	ns	
	t_D (IT_3)	モード3のときのデーター致からの遅延		420		380	ns	

信号	記号	パラメータ	Z-80 PIO 最小値	Z-80 PIO 最大値	Z-80A PIO 最小値	Z-80A PIO 最大値	単位	測定条件
A_0—A_7, B_0—B_7	t_S (PD)	モード1のときの$\overline{STB}$の立ち上がりに対するセットアップ時間	260		230		ns	C_L=50pF
	t_{DS} (PD)	モード2のときの$\overline{STB}$の立ち下がりに対するセットアップ時間		230〔4〕		210〔4〕	ns	
	t_F (PD)	モード2のときの$\overline{STB}$の立ち上がりからポート・バス・フロートまでの遅延		200		180	ns	
	t_{DI} (PD)	モード0のときの書き込みサイクルの$\overline{IORQ}$の立ち上がりからポート出力確定までの遅延		200〔4〕		180〔4〕	ns	
$\overline{A\ STB}$ $\overline{B\ STB}$	t_W(ST)	$\overline{STB}$のパルス幅（“L”）	150〔3〕		150〔3〕		ns	
A RDY, B RDY	t_{DH} (RY)	$\overline{IORQ}$の立ち上がりからの応答時間		t_c+460〔4〕		t_c+410〔4〕	ns	C_L=50pF
	t_{DL} (RY)	$\overline{STB}$の立ち上がりからの応答時間		t_c+400〔4〕		t_c+360〔4〕	ns	

注　〔1〕$t_C = t_W(\Phi H) + t_W(\Phi L) + t_r + t_f$

〔2〕負荷容量の50pF増加につき、遅延は10ns増加します。負荷容量の最大値は200pFです。

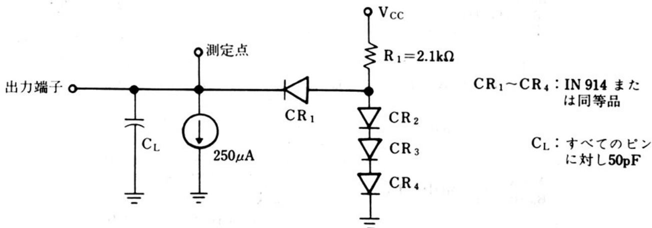
〔3〕モード2のときは、$t_W(ST) > t_S(PD)$となります。

〔4〕負荷容量の10pF増加につき、遅延は2ns増加します。負荷容量の最大値は100pFです。

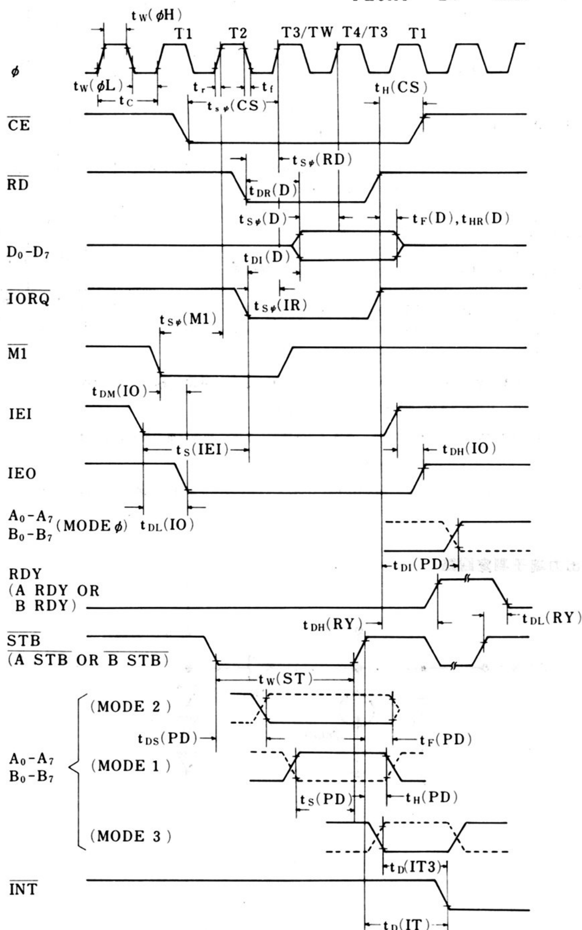
〔注1〕デージー・チェーンがN段ある場合

$2.5t_C > (N-2)\,t_{DL}(IO) + t_{DM}(IO) + t_S(IEI)$ + TTLバッファー遅延を満たさなければなりません。

出力端子測定回路



ACタイミング図


"H" "L"
CLOCK 4.2V 0.8V
OUTPUT 2.0V 0.8V
INPUT 2.0V 0.8V
FLOAT ⊿V +0.5V
tW(φH)
T1
T2
T3/TW
T4/T3
T1
φ
tW(φL)
tC
tr
tf
tH(CS)
tsφ(CS)
CE
tSφ(RD)
RD
tDR(D)
tSφ(D)
tF(D), tHR(D)
D0-D7
tDI(D)
IORQ
tSφ(IR)
tSφ(M1)
M1
tDM(IO)
IEI
tS(IEI)
tDH(IO)
IEO
A0-A7
B0-B7 (MODE φ)
tDL(IO)
tDI(PD)
RDY
(A RDY OR
B RDY)
tDL(RY)
tDH(RY)
STB
(A STB OR B STB)
tW(ST)
(MODE 2)
tDS(PD)
tF(PD)
A0-A7
B0-B7
(MODE 1)
tS(PD)
tH(PD)
(MODE 3)
tD(IT3)
INT
tD(IT)

（3） CTC

絶対最大定格

項　　目	記 号	定　　格	単位
入力電圧	V_{IN}	−0.3〜＋ 7	V
出力電圧	V_{OUT}	−0.3〜＋ 7	V
動作温度	T_{opr}	0〜＋ 70	℃
保存温度	T_{stg}	−65〜＋150	℃

電気的特性

DC 特性

（T_a = 0℃〜＋70℃，V_{CC} = ＋5V±5％）

記 号	項　　目	最小値	最大値	単位	測定条件
V_{ILC}	クロック"L"入力電圧	−0.3	0.45	V	
V_{IHC}	クロック"H"入力電圧	V_{CC}−0.6	V_{CC}+0.3	V	
V_{IL}	"L"入力電圧	−0.3	0.8	V	
V_{IH}	"H"入力電圧	2.0	V_{CC}	V	
V_{OL}	"L"出力電圧		0.4	V	I_{OL} = 2mA
V_{OH}	"H"出力電圧	2.4		V	I_{OH} = −250μA
I_{CC}	消費電流		120	mA	t_C = 400ns
I_{LI}	入力リーク電流		10	μA	V_{IN} = 0V〜V_{CC}
I_{LOH}	トライステート出力リーク電流		10*	μA	V_{OUT} = 2.4V〜V_{CC}
I_{LOL}	トライステート出力リーク電流		−10*	μA	V_{OUT} = 0.4V
I_{OHD}	ダーリントン駆動電流	−1.5*		mA	V_{OH} = 1.5V ZC／TO_0〜ZC／TO_2 に適用

＊流入電流を正、流出電流を負とします。

端子容量

（T_a = ＋25℃，f = 1MHz）

記 号	項　　目	最大値	単 位	測定条件
CΦ	クロック入力容量	25	pF	被測定端子以外のすべての端子は接地
C_{IN}	入力容量	5	pF	
C_{OUT}	出力容量	10	pF	

AC 特性

(Ta＝0℃～＋70℃、V_{CC}＝＋5V±5％)

信号	記号	パラメータ	Z-80 CTC 最小値	Z-80 CTC 最大値	Z-80A CTC 最小値	Z-80A CTC 最大値	単位	備考
Φ	t_C	クロック周期	400	〔1〕	250	〔1〕	ns	
	$t_W(\Phi H)$	クロック・パルス幅("H")	170	2000	105	2000	ns	
	$t_W(\Phi L)$	クロック・パルス幅("L")	170	2000	105	2000	ns	
	t_r, t_f	クロック立ち上がり・立ち下がり時間		30		30	ns	
	t_H	ホールド時間	0		0		ns	
CS, $\overline{CE}$	$t_{S\Phi}(CS)$	読み出し、または書き込みサイクルの制御信号のセットアップ時間	160		60		ns	
D_0-D_7	$t_{DR}(D)$	$\overline{RD}$の立ち下がりからデータ出力までの遅延		480		380	ns	〔2〕
	$t_{S\Phi}(D)$	書き込み、またはM1サイクルのデータのセットアップ時間	60		50		ns	
	$t_{DI}(D)$	INTAサイクルの$\overline{IORQ}$の立ち下がりからデータ出力までの遅延		340		160	ns	〔2〕
	$t_F(D)$	$\overline{RD}$の立ち上がりから出力バッファ・フロートまでの遅延		230		110	ns	
IEI	$t_S(IEI)$	INTAサイクルの$\overline{IORQ}$の立ち下がりに対するセットアップ時間	200		140		ns	
IEO	$t_{DH}(IO)$	IEIの立ち上がりからの遅延		220		160	ns	〔3〕
	$t_{DL}(IO)$	IEIの立ち下がりからの遅延		190		130	ns	〔3〕
	$t_{DM}(IO)$	$\overline{M1}$の立ち下がりからの遅延(M1サイクルの直前で割り込みが発生したとき)		300		190	ns	〔3〕
$\overline{IORQ}$	$t_{S\Phi}(IR)$	読み出し、または書き込みサイクルのセットアップ時間	250		115		ns	
$\overline{M1}$	$t_{S\Phi}(M1)$	INTA、またはM1サイクルのセットアップ時間	210		90		ns	
$\overline{RD}$	$t_{S\Phi}(RD)$	読み出し、またはM1サイクルのセットアップ時間	240		115		ns	
$\overline{INT}$	$t_{DCK}(IT)$	CLK/TRGの立ち上がりからの遅延		$2t_C(\Phi)$ +200		$2t_C(\Phi)$ +140	ns	カウンタ・モード
	$t_{D\Phi}(IT)$	Φの立ち上がりからの遅延		$t_C(\Phi)$ +200		$2t_C(\Phi)$ +140	ns	タイマ・モード
CLK/TRG 0−3	$t_C(CK)$	カウンタ・クロック周期	$2t_C(\Phi)$		$2t_C(\Phi)$		ns	カウンタ・モード
	$t_r(CK/TR)$ $t_f(CK/TR)$	カウンタ・クロックおよびトリガの立ち上がり・立ち下がり時間		50		30	ns	
	$t_S(CK)$	即時カウントに要するクロックのセットアップ時間	210		130		ns	カウンタ・モード
	$t_S(TR)$	プリスケーラの即時起動に要するトリガのセットアップ時間	210		130		ns	タイマ・モード
	$t_W(CTH)$	カウンタ・クロックおよびトリガのパルス幅("H")	200		120		ns	カウンタ・モードおよびタイマ・モード
	$t_W(CTL)$	カウンタ・クロックおよびトリガのパルス幅("L")	200		120		ns	
ZC/TO 0−2	$t_{DH}(ZC)$	Φの立ち上がりからZC/TO="H"までの遅延		190		120	ns	カウンタ・モードおよびタイマ・モード
	$t_{DL}(ZC)$	Φの立ち下がりからZC/TO="L"までの遅延		190		120	ns	

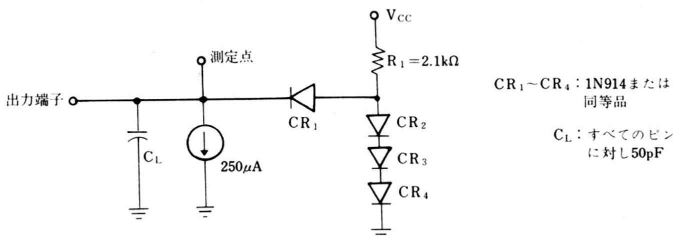
注〔1〕 $t_C = t_W(\Phi_H) + t_W(\Phi_L) + t_r + t_f$

〔2〕 負荷容量の50pF 増加につき、遅延は10ns増加します。負荷容量の最大値は、データ・バスが200pF であり、他は100pF です。

〔3〕 負荷容量の10pF 増加につき遅延は 2 ns増加します。負荷容量の最大値は100pF です。

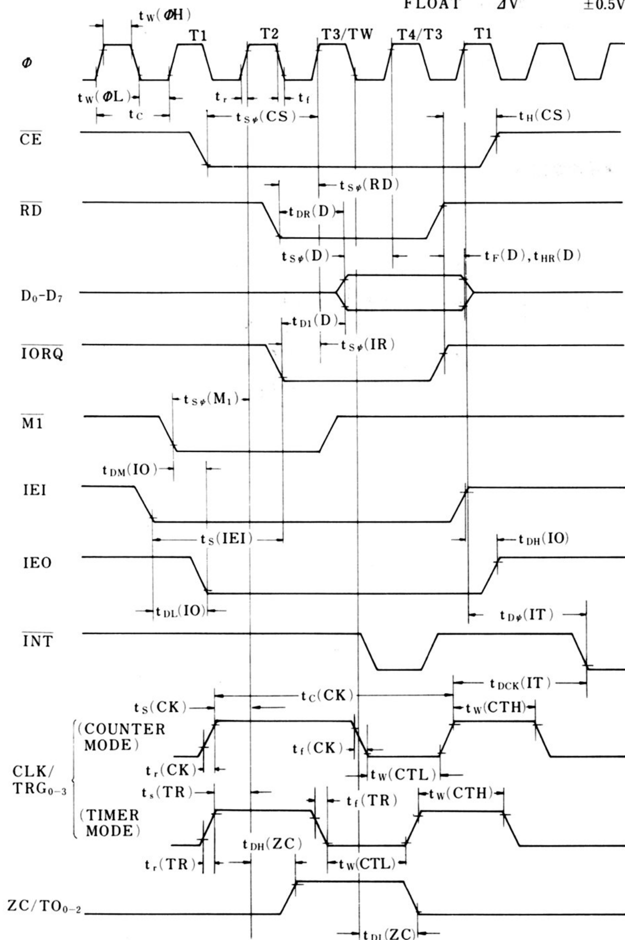
〔4〕 $\overline{\text{RESET}}$の入力幅は最低3クロック・サイクル必要です。

出力端子測定回路



AC タイミング図

	"H"	"L"
CLOCK	V_{CC}-0.6V	0.45V
OUTPUT	2.0V	0.8 V
INPUT	2.0V	0.8V
FLOAT	ΔV	±0.5V



著者略歴

昭和 47 年　東京電機大学工学部
　　　　　　電子工学科卒
現　　在　シャープ株式会社
　　　　　　電子部品営業本部

図解 マイクロコンピュータ
Z－80 の使い方

昭和56年 5 月20日　第1版第1刷発行
昭和59年 3 月30日　第1版第12刷発行

OHM·OHM·OHM
著者承認
検印省略

著　者　横田英一（よこたえいいち）

発行者　株式会社 オーム社
　　　　代表者 種田則一

発行所　株式会社 オーム社
　　　　郵便番号 101
　　　　東京都千代田区神田錦町 3－1
　　　　振　替　東京 6－20018
　　　　電　話　03(233)0641(大代)

組版　緑新社　　印刷　秀好堂印刷　　製本　大進堂

Printed in Japan

落丁・乱丁本はお取替えいたします

ISBN 4－274－07072－7

マイコン入門心得帖	平松・森本共著	四六判	**定価 1200円**
続・マイコン入門心得帖	平松・森本共著	四六判	**定価 1200円**
マイコン実験と工作マニュアル	北川一雄著	A5W	**定価 1900円**
続・マイコン実験と工作マニュアル	北川一雄著	A5W	**定価 2300円**
図解 初めてマイコンを学ぶ人のために	大原茂之著	A5判	**定価 1400円**
だれにもわかる マイコンの考え方・使い方	大條・吹浦共著	A5判	**定価 1700円**
図解マイコンの基礎知識	矢田光治著	A5判	**定価 2300円**
図解マイコンの使い方	小牧・大條共著	A5判	**定価 1400円**
図解マイコンのプログラム	平松・守屋・斉藤共著	A5判	**定価 1900円**
図解 マイコンのためのBASIC入門	小牧・大原共著	A5判	**定価 1500円**
図解 マイコンのためのアセンブラ入門	大原・倉田共著	A5判	**定価 1500円**
図解マイコンのための オペレーティングシステム入門	大原・倉田共著	A5判	**定価 1700円**
図解マイコンのための インタフェース入門	大原・北沢共著	A5判	**定価 1700円**
図解 マイコンのインタフェース	平松・斉藤共著	A5判	**定価 1800円**
ミニコン・マイコン入門	藤井・桑原共編	A5判	**定価 2000円**
マイクロコンピュータ入門	国分・藤井共著	A5判	**定価 1800円**
マイクロコンピュータの基礎	矢田光治著	A5判	**定価 1500円**
産業用マイクロコンピュータの基礎と応用	正田・泥堂・中島・松本共編	A5判	**定価 2300円**
制御用マイコンの作り方・使い方	北川一雄著	B5判	**定価 2900円**
制御用マイコンのプログラミング	北川一雄著	B5判	**定価 3900円**
計測自動化のための マイコン標準インタフェース	脇 英世著	A5判	**定価 2300円**
マイクロコンピュータの計測と制御への応用	岡田・富塚訳	B5判	**定価 3400円**
エンジニアのための 絵ときマイクロコンピュータ	吉本久泰著	A5判	**定価 1900円**
先生のためのマイコン教室	末武国弘監修	B5判	**定価 2300円**
家電技術者のための マイコンの使い方	阿部・金田・山下共著	A5判	**定価 2000円**
図解マイクロコンピュータ Z-80の使い方	横田英一著	A5判	**定価 1700円**
図解マイクロコンピュータ 続 Z-80の使い方	横田英一著	B5判	**定価 2800円**
図解マイクロコンピュータ Z-80アセンブラプログラミング入門	湯田・伊藤共著	A5判	**定価 2000円**
マイクロコンピュータ MC 6809の考え方	曽和将容著	A5判	**定価 2300円**
図解 16ビットマイクロコンピュータ 8086の使い方	井出裕巳著	A5判	**定価 1900円**
図解 PC-8801 N_{88}-BASIC入門	佐藤達男著	A5判	**定価 1900円**
マイクロコンピュータのための PL/M入門	小牧・大條・佐々木・坂本共著	A5判	**定価 1800円**
ワンチップマイコン入門	国分・浅見・野村共著	A5判	**定価 1700円**

定価 1700円